结合重点案例　深度剖析互联网中的流处理技术

结合实际应用　重点介绍流量大数据分析的问题

互联网流量大数据工程

陈　震　黄文良　曹军威　编著

Big Data Engineering on Internet Traffic

清华大学出版社

互联网流量大数据工程

陈 震 黄文良 曹军威 编著

清华大学出版社
北 京

内容简介

本书在全面介绍互联网流量大数据获取、索引、存储和分析等基本知识的基础上，着重介绍基于多级过滤器和空时 Bloom Filter 的流量测量技术、基于网包调度的流量管理技术、基于模式匹配的攻击检测技术、基于流式计算的实时攻击检测方法、基于流量分析的互联化安全软件的行为分析和基于大数据平台的互联网流量存储与处理的方法，并通过中国联通研究院的移动互联网监控平台和互联网企业流量大数据平台的实际例子，说明这些技术的具体应用，全书提供了大量应用实例。

全书共分 8 章：第 1 章介绍互联网流量大数据背景，包括移动互联网发展态势。第 2 章介绍互联网流量测量与带宽管理，着重介绍高速网络流量测量与网包调度算法及应用。第 3 章介绍互联网流量归档与查询，该技术是流量大数据运营的前提。第 4 章介绍互联网流量大数据存储，大数据存储是流量大数据运营的基础，当前流行的大数据存储架构为 HDFS/HBase 技术。第 5 章介绍互联网流量攻击检测的流扫描和多模匹配 HBM 算法等关键技术。当前网络攻击形式更加多样化，如何高效深度分析网流内容，查找网络攻击源头是流量大数据的一个创新应用。第 6 章介绍互联网流量大数据的网络攻击检测，网络攻击实时检测是保障互联网服务安全的重要手段，介绍了互联网企业在保障业务安全中的攻击检测手段。第 7 章介绍互联网化软件流量行为分析。互联化终端软件是通过云计算平台部署的，能够充分利用互联网基础服务。在改善用户体验的同时，也会带来用户隐私泄露的隐患。第 8 章回顾了前 7 章内容，总结全书。

本书适合作为高等院校计算机系统架构、计算机网络及安全专业高年级本科生、研究生的教材，同时可供对计算机网络比较熟悉并且对大数据平台有所了解的开发人员、广大科技工作者和研究人员参考。

图书在版编目(CIP)数据

互联网流量大数据工程/陈震，黄文良，曹军威编著. --北京：清华大学出版社，2014
ISBN 978-7-302-36083-4

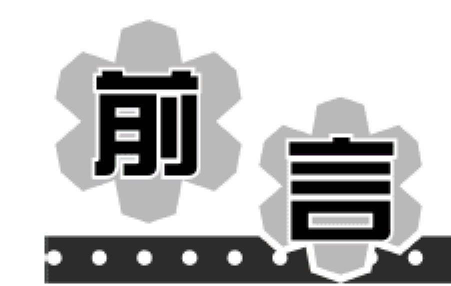
Ⅰ. ①互…　Ⅱ. ①陈…　②黄…　③曹…　Ⅲ. ①互联网络—数据处理　Ⅳ. ①TP393.4

中国版本图书馆 CIP 数据核字(2014)第 069674 号

责任编辑：白立军
封面设计：
责任校对：李建庄
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×230mm　　**印　　张**：12.5　　**字　　数**：306 千字
版　　次：2014 年 7 月第 1 版　　**印　　次**：2014 年 7 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：057620-01



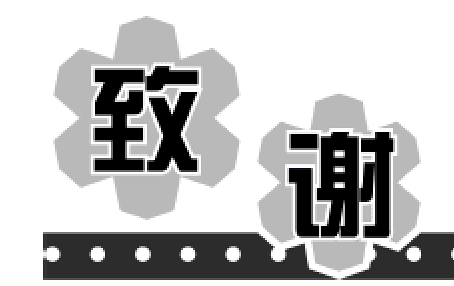
随着互联网应用的普及和渗透,宽带移动无线网络的大规模商用,整个互联网流量总量高速增长。移动用户可以通过任何设备,在任何地方,获取任何想要的内容,产生了大规模的移动流量数据。据思科公司报告[1]预言,互联网的流量数据在2011到2016年之间将增长4倍,并于2016年达到1.3ZB($1ZB=10^{21}B$)。据中国联通公司统计,2012—2013年,移动用户流量的复合增长率为135%,3G月均流量增长最高。而对于一家大型的互联网公司(如百度、淘宝、腾讯等),在日常运营中生成和累积的用户行为数据是相当庞大的,往往达到TB($1TB=10^{12}B$)级甚至PB($1PB=10^{15}B$)级的数据。如此庞大的数据流量的管理是一个具有挑战性的问题,需要一整套大数据存储、处理和分析平台。

本书详细探讨了互联网流量大数据的获取、存储、分析和处理的关键技术,如网包在线抓取,流量档案化,索引及索引压缩算法,基于HDFS与HBase的分布式可扩展并行存储与分析架构,应用于攻击检测的模式匹配算法,面向流式与实时计算的大数据架构及处理方法。在此基础上,对流量大数据进行深入分析挖掘,如用户行为分析、攻击检测和应用软件行为分析等。同时,互联网流量测量也是运营商管理网络的基本手段,诊断可能存在的性能瓶颈,发现网络链路故障。通过流量带宽管理手段,对不同优先级的网络流量进行分级管理,实现网络的最优化运营。对互联网流量大数据分析,可以实时在网络上监控发生的每一个请求、服务、交易;对途经流量进行深层次报文分析(DPI),重组应用层有效信息,双向匹配请求/服务并记录,从而尽最大可能发现各种安全攻击,并进行实时阻断。中国联通公司进一步提出了流量大数据产业经济方案,基于流量大数据的生态系统产业经济建设的方案,联合多个合作方,包括基础框架服务商ISP、运营商的大数据服务商、大数据分析与挖掘企业和内容与客户端应用商。

本书内容经过10多年实际科研项目工作积累使用,选取的内容力求丰富全面,基本概念的讲解细致,深入浅出。该书作者长时间在互联网网络安全领域从事研究,熟悉分布式系统、互联网安全和软件可信性分析,尤其对该书投入巨大精力,尽可能让读者在较短时间内全面把握互联网流量的获取和分析处理,并应用于实际工作中。

作　者

2014年5月



本书得到国家973项目(未来互联网体系结构与机制研究)和国家自然基金项目(下一代互联网安全与隐私关键性技术的研究)的支持。

本书的内容是研究工作的集成，旨在系统化反映整体的研究进展，主要概括了研究组自2004年以来的研究成果。其中许多段落直接来自同学的博士和硕士论文与调研报告，他们是韩阜业、梁勇、阮凌云、倪嘉、阮东华和蒙金莉等。

以下同学参与了本书整理工作，他们是董文宇(第2章和第4章)、李航(第3章)、曹彬(第7章)、谢珍真等同学参与了本书的编写工作。

感谢清华大学计算机系网络安全及性能评价实验室QoSlab的NP组的倪嘉、阮东华、岳峣、谭章熹、郑波和洪孙安(Peter Ungsunan)等在网络处理器上的科研工作。

感谢清华大学信息技术研究院网络安全实验室NSLAB的CORS组的周晋博士、唐力博士、张辉博士、张志明博士、李光华、杜剑和郭瑛等在覆盖网的科研工作。

感谢清华信息技术国家实验室李军教授、汪东升教授、尹浩教授、邢春晓教授和任丰原教授等的支持和鼓励。


目
录

第1章

互联网流量大数据背景

当前的互联网体系结构是基于终端间 TCP/IP 的点到点通信架构，随着互联网尤其是移动互联网的发展，连接的上网终端设备已经达到数十亿台。Cisco 公司的报告表明，互联网流量 2012 年每月数据量为 44 艾字节(EB，Exabyte)，是 2009 年每月流量的 3 倍，流量大数据时代已经来临。互联网流量大数据体现了大数据的典型 4V 特征：流量数据的大小(Volume)、流量数据的类别(Variety)、流量数据的速度(Velocity)和流量数据的价值(Value)。互联网流量数据量大、协议类别多、产生速度高、蕴含价值大，对流量大数据的分析处理是产生价值的重要途径。对于国家监管部门而言，流量大数据具有国防公共安全价值；对运营商和商业用户而言，互联网用户的行为和内容获取，具有重要的商业价值。因此有必要对网络流量进行记录和深入分析。

1.1 网络流量记录与分析

随着互联网的飞速发展，人们的工作、学习和生活均已经和网络密切相关。对网络的内容和运行状况监控，保证网络健康正常地运行已成为一项重要工作。在中国“十二五”规划纲要中，已经明确提出要“加强网络与信息安全保障”，充分体现了国家对信息安全的重视。

而网络的开放性造成了网络攻击的普遍性。在网络链路方面，网络中某个节点的错误配置可能会给整个网络带来灾难性的后果；网络攻击会造成链路的阻塞，服务器的崩溃，甚至是局部网络通信的中断。在网络内容方面，人们可以在各个地方上传不良信息，进行非法活动，给其他互联网的使用者带来不好的思想、精神、经济等方面的影响和损失。由于这些行为常常不能在发生时被发现，因此需要对网络流量进行记录，以供后期进行研究、分析和举证。

ISP(Internet Service Provider)管理和运行大型的高速网络，链路速度在 10～100Gbps 级，即每秒产生的数据量为 1～10Gb 量级，如果要进行流量的记录，代价非常高。

这样的设备和技术的价格也是一般用户无法承受的。同时,互联网的发展使许多公司拥有自己的企业网络,也有许多公司将自己的服务器托管到 IDC(Internet Data Center)运行。这样的局部网络的数目非常庞大,其网络带宽一般在 100～1000Mbps。管理这些局部的网络也是一个非常现实的问题,为这样的网络设计和实现廉价的网络流量记录工具具有很大的应用前景。

1.2 网络流量研究现状

通过对以往研究的调查可以发现,流量信息的存储和记录已有不少研究成果。在本文中将以往的工作分两类分别介绍,它们分别是网流信息的记录与查询、网络流量的记录和查询。两者名字看上去很相似,但是却对处理性能要求有截然不同的差距。前者是指对网络中的流信息的记录与查询,要记录和归档的内容是描述一个网流的信息,比如常用的五元组信息(源 IP 地址,目标 IP 地址,源端口,目标端口,传输层协议),网流的大小,产生时间,持续时间;但不对网络流量的内容作具体的记录。而后者要求将网络中实际传输的网包记录下来。

1.2.1 网流信息的归档与查询

在过去的 10 年中,由于系统的硬件发展所限,特别是存储设备,如硬盘大小限制,记录实际的网络流量是件十分困难的事情。因此,对网络流量的检测与统计靠的是对网络流量的抽象信息。这些抽象信息能满足对网络中流量进行统计、监测和事件回溯的基本需求。如今广泛使用的各种网络监测、管理软件都是依靠这样的信息抽象方式来表达网络运行的状况。

NetFlow[1] 和 sFlow[2] 是现行的在业界广泛使用的对网流进行描述的标准。它们的信息来源于网络中的探测点(如路由器)向外界发送的对网络实时流量的分析与统计得到流信息。之前的研究主要是建立一个这样的流信息的收集、存储、归档和检索系统。使得这个系统能很好地服务于网络监控程序。

由于流量信息具有量大、变化快(每秒新建流数量大)的特点,如果要从探测点获取精确的信息就要求探测点为每个流做细粒度的统计,而这会使得网络中探测点的负载增加,从而降低这些探测点的本职工作效率(如路由器的转发效率)。因此,探测节点经常利用采样的方法降低工作负载,但这又导致了流信息的不准确。

网流信息归档与查询系统要求收集来自网络中探测节点的流信息,存储到本地,并向外提供查询接口。换言之,网流信息的归档与查询系统就是要建立一个网流信息的数据库。数据库中的每条信息是收集自网络中探测节点的一条网流描述。数据库的输入是查询语句和不断增加的网流信息条目,输出是想要查询的相关的网流信息。

总结相关的研究和工作，可以把存储的网流信息分为三类。

1. 原始文件

使用原始文件的好处是，可以以最快速度记录收集到的流信息，并且这种方案将流的记录和分析分开成两个工作。这样做的缺点是某些流量信息记录下来之后，要经过一段时间才会被分析模块处理，也就是说，对某一条在 t 时刻发生的事件，要在 $t+\Delta t$ 时刻才能被查询，而在 $(t, t+\Delta t)$ 时间段内，查不到其实已经收集到的信息。因此采用原始文件的方案降低了查询的实时性。

使用原始文件的另一个缺点是，由于没有对信息做任何索引，检索时将不得不在原始文件中逐条遍历，这使得查询性能变得不可容忍。但是这个缺点可以通过一些其他手段缓解，比如根据时间把不同时段的流存储到不同的文件。但当用户不使用时间区域限制或将时间范围设置得很大时，这种手段又失效了。

属于这类的工作有 nfdump[3] 和 OSU flow-tools[4]。

2. 通用数据库

使用通用数据库的好处主要有：安全稳定的数据库支持，无须自己开发，灵活方便的 SQL 语句查询。通用的关系型数据库被广泛用于各种类型数据的存取。但是对于某一种特殊的应用，不一定能取得最理想的效果，因为它没有利用该应用的数据特点。

以往的研究表明，使用通用关系型数据库进行存储流信息，其插入、查询的效率甚至比使用原始文件的效率还要低，并且还要占用更大空间[5]。

属于这类的工作有 Neye[6]，Combining[7]。

3. 专用数据库

为专门的数据设计专用的数据库才能根据数据的特点对数据的存储和查找在时间和空间性能方面做优化。Gigascope[8] 建造了高性能的网流信息数据库，在一台普通的主频为 2.4GHz 的双核服务器上，每秒能处理 1 200 000 个包的信息，并且自己实现了方便灵活的、仿 SQL 语句的查询语法 GSQL。Tribeca[9] 是 10 多年前提出的一个方案，其中明确指出了不使用通用关系数据库的几大原因：流状的数据不需要经过更改，但要求快速顺序访问；通用关系数据库不能很好地支持统计上的常用计算，如计算时间间隔；空间消耗比较大。Tribeca 提出了自己的查询系统和相应的查询语句。Collecting[5] 提出了使用 Bitmap 索引数据库对流信息进行储存和查询的方法，取得了比通用关系型数据库更好的性能。

1.2.2　网络流量的归档与查询

由于硬件的不断发展，TB 级的硬盘已经开始普及，价格也下降到了可以接受的范围。同时计算性能的发展使得完全记录网络流量，而不仅仅记录一些统计信息成为可能。

过去关于网络流量的归档与查询系统的设计与研究，可以分为两个方面：系统级方案和应用级方案。

1. 系统级方案

系统级方案是指为了实现网络流量的归档与查询，从操作系统这一层面提出的解决方案。这样的方案充分利用机器的硬件性能。这方面工作十分突出的是 Hyperion[10]。Hyperion 设计了一个支持高速的网流存储、索引和在线查询的系统。Hyperion 系统能够在一台普通计算机上，每秒记录大于 1 000 000 个网包，同时支持在保证查询性能的同时每秒进行 200 000 个网包的索引。Hyperion 实现的是面向流的文件系统，使磁盘能保证连续读写，从而达到最高的写速率。同时 Hyperion 也提出了自己的多层级索引方案。

2. 应用级方案

应用级方案是指在常用的操作系统上构建网络流量归档查询系统的解决方案。这样的方案使系统能够与其他功能很好地兼容。如果采用系统级方案，意味着设备只能用作唯一用途，如果要附加即使很小的一个功能，也需要开发适应该系统的新的软件。而使用应用级方案，能够和其他功能并存，也便于系统的管理。TM[11][12]是应用级方案的代表。它的设计思想是设计高效的网络流量的记录和查询工具。TM 提出了对流截断基本不会损失网络流量中蕴藏的信息量的论断，并且给出了很好的验证。而对流进行截断操作大大降低了要处理的流量，使得系统能够胜任 Gbps 级网络的流量归档查询工作。

1.2.3 对研究现状的分析

无论是网流信息归档，还是网流归档，两者的数据形式和特点都是相似的：都是海量的、流状的、不再更改的数据。系统的设计都是要提高对这样的数据进行存储和查找的性能，因此可以放在一起做比较。总结已有的研究，根据数据存储和索引的方式进行分类，得到表 1.1。

表 1.1 不同方案的对比表格

方　　案	原始文件	通用数据库	专用数据库或系统
优点	存储速度快	查询语句丰富	高效
缺点	查询实时性	效率较低	
代表系统	TM nfdump OSU flow-tools	Neye Combining	Gigascope Hyperion Bitmap Database
开发难度及说明	中 需要设计和实现索引模块加快查询速率	低 数据的储存和索引都不再需要编写	高 需要设计和实现专用的数据库

通过表 1.1 可以发现,只有构建专用的数据库或系统才能取得最好的存储和查询性能。但是要完成 100Mbps～1Gbps 网流的存储,要求每秒记录 12.5MB～125MB 的数据量,而普通硬盘的写瞬时速率最高在 60MBps 左右,写大文件的速率在 30～40MBps,写小文件的速率远小于 10MBps。因此完成这样的网络环境下的网流存储,普通硬盘很难支持。同时,1Mbps 的流量一天产生的数据量为 10.8GB,100Mbps 的流量一天产生的数据量超过 1TB,这样的数据量在 1TB 硬盘普及的今天,也是无法接受的。同时,100MBps 意味着每秒要处理 0.1～1M 个网包,这对计算机处理器的性能也是一个极大的挑战。因此在一台普通计算机上将这样级别的流量完完整整地存下来,是非常困难的一件事情。因此人们不禁要探讨,真的需要将所有的网包都记录下来吗?网流中什么信息是有效的?什么信息又是可以舍弃的呢?

但是因为存储空间的限制,如果对网流中的信息进行舍弃,那么舍弃的极限,就是回退为流信息的记录。我们想要将网络流量记录下来是因为不满足于流信息的记录所提供的信息量。

流信息只是简单地对网络中的一个流进行一个概括,从何处来,往何处去,什么时间开始,持续了多久。而实际流量中却蕴藏了各层协议的信息,尤其是应用层的数据。在事件发生后的回溯调查过程中,仅仅掌握流信息很难进行细致的考究。而如果拥有所有信息,人们能在事后对安全事件做更加深入的调查,因此记录实际流量是很有价值的。

Time Machine 系统设计提出了网络流量在记录时可以截断,而同时保留绝大部分信息的方法。如果采用流截断的方法,对于每个流只存储流开始的一部分数据,而舍弃后面的网包,大大降低了系统对磁盘读写性能、存储空间和运算能力的要求。

1.3　互联网流量大数据平台

以上只是对单条链路或者是单个企业网络的流量归档与查询,如果要对运营商的网络整体流量归档与分析,面对的是 PB 级流量大小,就必须采用大数据的平台架构和分析方法。

以中国联通公司为例,对于移动用户上网记录而言,当前每日的上网记录数据生成量超过 300 亿条,每日数据量约 8.4TB。每月的上网记录数达到万亿条,存储容量达到 PB 级。并且移动互联网用户访问流量在快速增长(大约每半年翻一番),由此引发的上网记录数据将进一步猛增。

基于 Hadoop 的分布式存储架构,利用面向列的 HBase 数据库,实现海量数据的高效检索和分析处理。该架构使用廉价的 PC 服务器和存储实现了系统的高可靠性,相比

IOE(IBM 小型机、Oracle 数据库、EMC2 高端存储)的传统方案,可以节省开销。

Hadoop 是以 Google 公司的 GFS 为样板的开源实现,已经成为事实的标准。Hadoop 2.0 YARN 更支持了实时计算,功能更加强大。

Hadoop 是一个分布式系统基础架构。Hadoop 实现了一个分布式文件系统(Hadoop Distributed File System,HDFS)。HDFS 有高容错性的特点,并且设计用来部署在低廉的(Low-cost)硬件上。而且它提供高传输率(High Throughput)来访问应用程序的数据,适合那些有超大数据集的应用。

图 1.1 描述了 Hadoop 各层系统之间的层次,其中 HBase(Hadoop Database)位于结构化存储层,Hadoop HDFS(Hadoop Distributed File System)为 HBase 提供了高可靠性的底层存储支持,Hadoop MapReduce 为 HBase 提供了高性能的计算能力,Zookeeper 为 HBase 提供了稳定协同服务和错误纠正机制。此外,Pig 和 Hive 为 HBase 提供了高层语言支持,使得在 HBase 上进行数据统计处理变得非常简单。

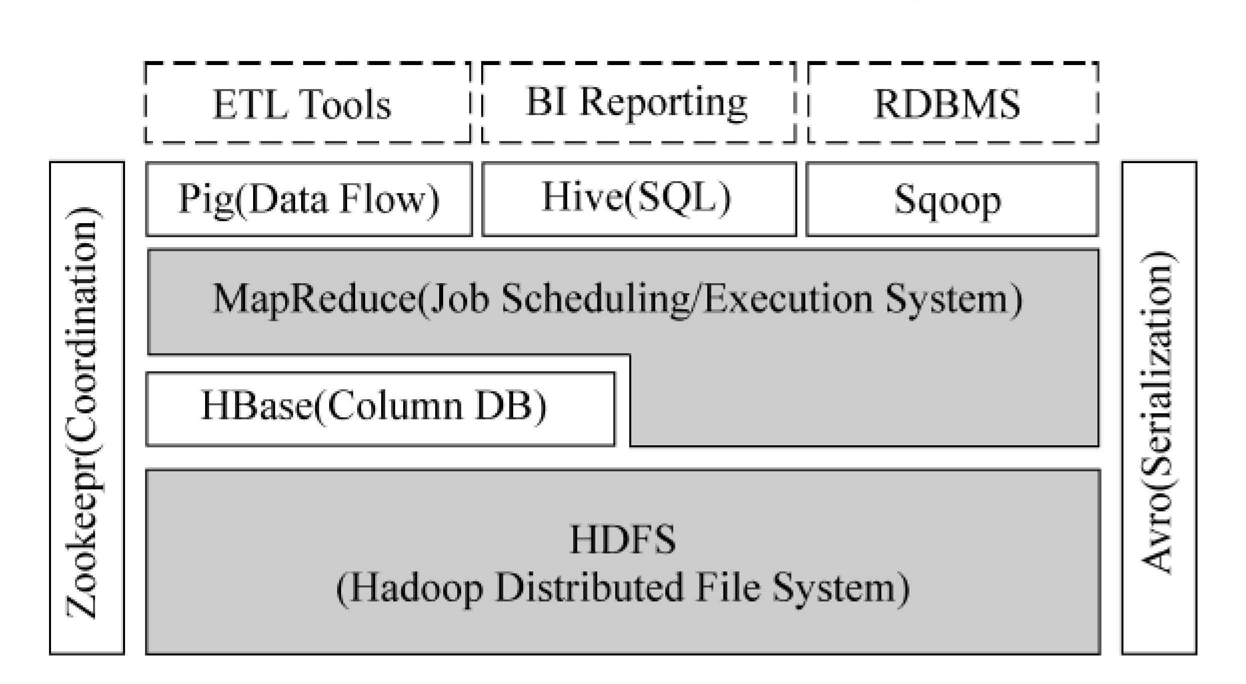


图 1.1 基于 Hadoop 的大数据系统层次

HBase 基于可以横向扩张的表存储特性,可以利用 HBase 技术在廉价 PC Server 上搭建起大规模结构化存储集群,通过不断增加廉价 PC Server 增加计算和存储能力,这样就可支持 Terabyte 到 Petabyte 级的海量数据存储和高速读写。基于 Hadoop 的 HBase 数据库系统如图 1.2 所示。

对图 1.2 中的主要部件介绍如下。

1. HMaster

HBase 中可以启动多个 HMaster,通过 Zookeeper 的 Master 选举机制保证总有一个 Master 运行,HMaster 在功能上主要负责 Table 和 Region 的管理工作。

2. HRegionServer

主要负责响应用户 I/O 请求,向 HDFS 中读写数据,是 HBase 中最核心的模块。

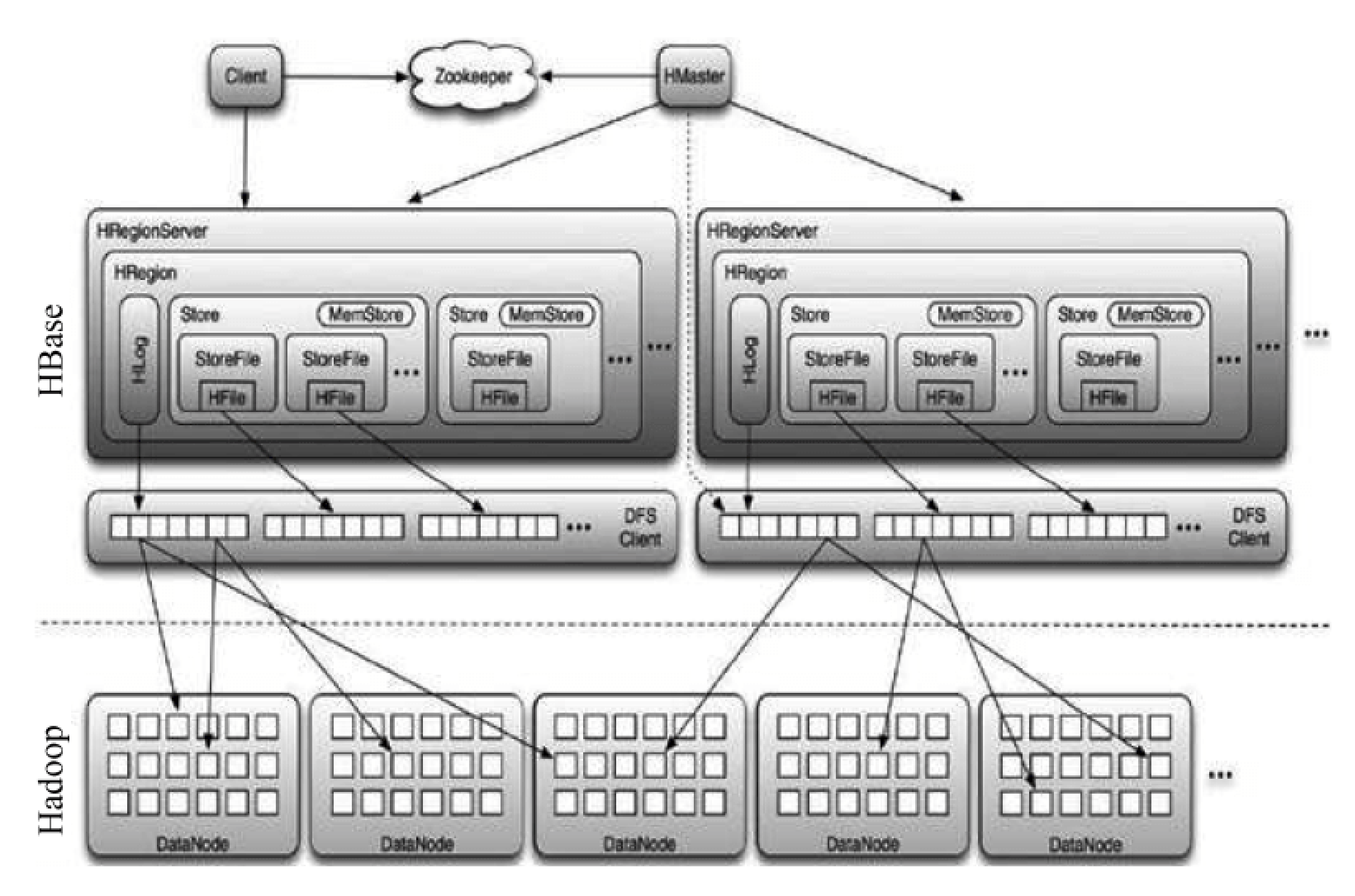


图 1.2　基于 Hadoop 的 Hbase 数据库系统

HRegionServer 内部管理了一系列 HRegion 对象，每个 HRegion 对应了 Table 中的一个 Region，HRegion 中由多个 HStore 组成。每个 HStore 对应了 Table 中的一个 Column Family 的存储。

3. Zookeeper

Zookeeper 中除了存储了-ROOT-表的地址和 HMaster 的地址，HRegionServer 会把自己注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个 HRegionServer 的健康状态。此外，Zookeeper 可以避免 HMaster 的单点问题。

4. Client

使用 HBase 的 RPC 机制与 HMaster 和 HRegionServer 进行通信，对于管理类操作，Client 与 HMaster 进行 RPC；对于数据读写类操作，Client 与 HRegionServer 进行 RPC。

为此，中国联通公司采用了基于开源的 Hadoop/HBase 作为上网记录的存储方案。基于 Hadoop/HBase 上网系统为中国联通公司客服人员提供上网记录快速查询服务，解决流量投诉问题；基于 MapReduce 和 Hadoop/HIVE 数据仓库技术，系统实现了统计报表和用户行为分析服务；基于 Hadoop/HBase，系统可以提供 IP 地址溯源服务。移动用户上网记录检索与应用分析系统如图 1.3 所示。

当前大数据实时分析平台是目前研究的热点，其中以 Twitter storm 为代表的流式计算是当前的一个趋势。

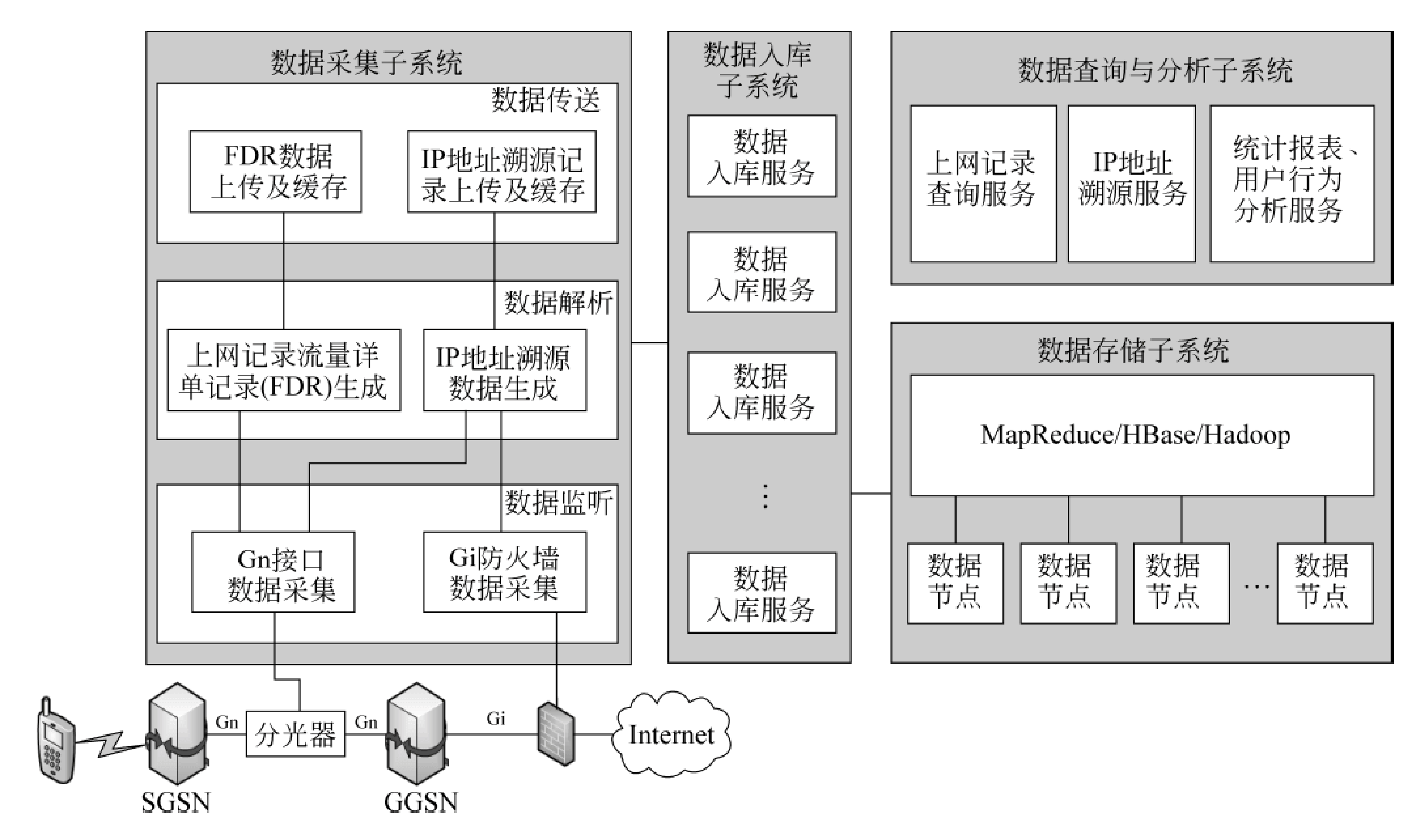


图 1.3　移动用户上网记录检索与应用分析系统

1.4　国际前沿进展

互联网流量大数据涉及多个计算机学科方向，包括计算机系统结构、计算机网络、数据库系统和人工智能等。其目标是要解决流量大数据的存储、索引查询和分析挖掘等重要科学工程问题。

国际流量监管与分析（TMA）会议[19]至今已经举办了 6 届，是关于网络流量监管与分析的正统会议。TMA 2012 会上提出的 TSDB[23]是一种压缩时间序列的方法，相比于 RRD 方法和 Reddis 数据库，具有检索快、存储空间小的优点。

国际测量大会（IMC）会议[20]也有流量分析的专栏，但是主要以网络测量和分析方法为主。RasterZip[22]是针对网流检索压缩的一种高效压缩格式和方法。

NET-FLi 系统[24]是超大规模数据库（VLDB'2012）会议上提出的针对网流查询与处理的数据库系统，采用 COMPAX 的位图压缩机制，用在线近邻敏感散列函数（oLSH）重排序流记录条目，实现在线索引压缩与档案化流记录。

1.5　小　　结

随着互联网流量的激增,对网络流量的记录和分析是一个挑战问题,传统通过流量记录和归档查询系统,已经满足不了运营管理整个 ISP 网络的需求。而大数据平台技术的兴起,为记录、处理和分析互联网流量提供了新的方法与手段,为充分挖掘流量大数据的价值奠定了良好的基础。

参 考 文 献

[1] Benoit Claise. NetFlow Services Export Version 9, RFC 3954, 2004.

[2] Peter Phaal, Sonia Panchen, and Neil McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed networks, RFC 3176, 2001.

[3] Peter Haag. Watch your Flows with NfSen and NfDump. 50th RIPE Meeting, 2005.

[4] Mark Fullmer, and Steve Romig. The OSU flowtools package and CISCO NetFlow logs. In Proceedings of the 2000 USENIX LISA Conference, 2000.

[5] Luca Deri, Valeria Lorenzetti, and Steve Mortimer. Collection and exploration of large data monitoring sets using bitmap databases. In Traffic Monitoring and Analysis, 2010.

[6] NEye. An Open Source NetFlow collector. http://neye.unsupported.info.

[7] John-Paul Navarro, Bill Nickless, and Linda Winkler. Combining Cisco NetFlow Exports with Relational Database Technology for Usage statistics, Intrusion Detection and Network Forensics. In Proceedings of the 14th Systems Administration Conference (LISA), 2000.

[8] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003.

[9] Mark Sullivan, and Andrew Heybey. A system for managing large databases of network traffic. In Proceedings of USENIX, 1998.

[10] Peter Desnoyers, and Prashant J. Shenoy. Hyperion: High Volume Stream Archival for Retrospective Querying. In Proceedings of 2007 USENIX Technical Conference, 2007.

[11] Stefan Kornexl, Vern Paxson, Holger Dreger, Anja Feldmann, and Robin Sommer. Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic. In Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, 2005.

[12] Gregor Maier, Robin Sommer, Holger Dreger, Anja Feldmann, Vern Paxson, and Fabian Schneider. Enriching Network Security Analysis with Time Travel. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 183~194.

[13] Kesheng Wu, Sean Ahern, E. Wes Bethel, Jacqueline Chen, Hank Childs, Estelle Cormier-

Michel，and Cameron Geddes et al. FastBit：Interactively Searching Massive Data. In Journal of Physics：Conference Series，2009，180(1)：12053.

[14] 范承工，周宝曜，刘伟. 大数据：战略，技术，实践[M]. 北京：电子工业出版社，2013.

[15] 刘鹏. 云计算(第 2 版)[M]. 北京：电子工业出版社，2011.

[16] 唐宏，秦润锋，范均伦. 开源云 OpenStack 技术指南[M]. 北京：科学出版社，2013.

[17] 陈伯龙，程志鹏，张杰. 云计算与 openstack[M]. 北京：电子工业出版社，2013.

[18] 刘军. Hadoop 大数据处理[M]. 北京：人民邮电出版社，2013.

[19] TMA 2014，The 6th Traffic Monitoring and Analysis Workshop. http://networks.eecs.qmul.ac.uk/news/tma-2014/.

[20] IMC2014：Internet Measurement Conference 2014. http://conferences.sigcomm.org/imc/2014/.

[21] Francesco Fusco，Michail Vlachos，and Xenofontas Dimitropoulos. RasterZip Compressing Streaming Network Monitoring Data with Support for Partial Decompression. In Proceedings of the 2012 ACM conference on Internet measurement conference，2012.

[22] Luca Deri，Simone Mainardi，and Francesco Fusco. tsdb：A compressed database for time series. In Traffic Monitoring and Analysis，2012.

[23] Francesco Fusco，Xenofontas Dimitropoulos，Michail Vlachos，and Luca Deri. pcapIndex：an index for network packet traces with legacy compatibility. ACM SIGCOMM Computer Communication Review，2012，42(1)：47～53.

[24] Francesco Fusco，Michail Vlachos，and Marc Ph Stoecklin. Real-time creation of bitmap indexes on streaming network data. The VLDB Journal-The International Journal on Very Large Data Bases，2012，21(3)：287～307.

[25] Francesco Fusco，Marc Ph Stoecklin，and Michail Vlachos. NET-FLi：on-the-fly compression，archiving and indexing of streaming network traffic. The VLDB Endowment (VLDB)，2010，3(1-2)：1382～1393.

[26] WenLiang huang，等. Mobile Internet Big data in China Union，Tsinghua Science and Technology vol. 19，issue 1，2014.

第2章

互联网流量测量与带宽管理

随着互联网用户的不断增加和用户需求的不断变化,互联网上的应用类型也在发生着日新月异的变化。各种类型的业务流量不断增长,网络的负载不断加重,互联网流量测量与管理的重要性日渐突出。

网络测量是网络行为学研究的主要手段。通过网络测量,掌握 Internet 行为规律是进行网络建模、网络规划、网络管理和网络安全等研究工作的重要前提。在当前 Internet 的带宽、拓扑和用户数的规模都爆炸式增长的情况下,流量测量遇到了很大的挑战,许多新的算法、硬件方案相继提出。测量算法除了最简单的原始记录算法,主要代表有采样算法、Multi-Stage Filter 算法和 Multi-Resolution Space Code Bloom Filter 算法等。

在掌握了网络测量技术的基础上,如何通过合适的流量管理技术,即网络应用协议的识别与带宽管理技术,来协调网络应用与安全、带宽增长与业务收益、网络扩容与用户体验之间的关系,是网络管理者与运营者亟待解决的重要挑战。

本章将介绍网络流量测量与管理的基本概念和关键技术,对主流的流量测量技术、算法和开源流量管理系统,以及商用流量管理系统进行介绍,并提出一种协同式的分布流量管理系统,分布式的部署流量管理子系统,通过分析控制中心协同检测和管理流量,可以有效地提高网络流量管理效率,提高未知网络协议的识别率,有效地检测和防止 DDoS 攻击。

2.1 流量测量概述

2.1.1 Internet 流量测量

流量测量通过记录网络上流(Flow)的流量值,回答"Who,Where,When,How,What"的问题,即什么用户在什么地址、什么时间,用什么方式,做了什么事情。

网络流量测量可以用于:计费(例如,按流量计费、针对用户计费等)、流量工程(根据

链路上的流量特征进行路由规划，以达到负载均衡，提高网络性能）和故障诊断（例如，根据流量异常发现网络路由存在的问题等）等。

流量测量的一个最重要的应用就是入侵检测系统（Intrusion Detection System，IDS）。众所周知，现有的 IPv4 网络是一个缺乏安全基础的网络，蠕虫爆发（2000 年以后的 RedCode、Slammer 等病毒都造成了巨大的损失）、黑客攻击（DDoS 攻击曾经造成数家大型网站无法运作）愈演愈烈。而根据流量测量的结果中某个 IP、端口或者 ICMP 报文的流量异常可以发现主机扫描、端口扫描等，判断并阻止网络入侵。

因此，进行网络流量测量的研究，对于了解网络特点，提高网络性能，阻止网络攻击有重要的意义。

2.1.2 流量测量的难点

当前网络发展的趋势是规模越来越大（在 Internet 骨干网上一小时内的流数目可达 1.7×10^6），带宽越来越宽（10Gbps 甚至更高）。大规模高速网络对流量测量提出了新的要求：能够线速处理到达的大量网包。这就必须研究新的算法和硬件平台。

2.1.3 流量测量的目的

本章首先比较了目前学术界已经提出的一些新的测量算法，分析这些算法的优劣，时间、空间复杂度等，从中选择适合高速网络的测量算法。

其次，利用通用多核平台进行流量测量。多核处理器采用了多内核并行结构，硬件多线程技术等适应网络网包的处理，是当前流量测量的主要平台。

目前利用流量测量进行入侵检测的手段，有考察 IP、端口流量异常，ICMP 目的地不可达报文数量异常，网络总流量的小波变换参数突变等。

此外，当前的研究热点还有利用网络自相似特性进行流量测量等。

2.1.4 网络测量与流量测量

流量测量是网络测量的一部分。网络测量是指遵照一定的方法和技术，利用软件和硬件工具来测试或验证表征网络性能的指标的一系列活动的总和。网络测量的对象除了流量以外，还包括网络拓扑、路由、带宽和延时等。

通过网络测量，掌握 Internet 的行为是网络建模、网络规划、网络管理和网络安全、新网络协议和网络应用设计等诸多研究工作的重要前提。不测量就无法理解网络，不测量就无法建立一个有效的模型。从研究的实质上看，网络测量可以把因特网从技术上升到科学，并且能够更好地指导应用。尤其在今天，Internet 的带宽、拓扑和用户数的规模都爆炸式增长，上述工作更是必不可少。

鉴于网络检测的重要性，IETF 组织已建立了 IPFIX（IP 流信息输出）工作组（关注 IP

流描述和信息导出)和 PSAMP[2] 工作组(关注网包采样),并提出了许多相关的 RFC,如 RFC 3917 (Requirements for IP Flow Information Export)、RFC 3955(Evaluation of Candidate Protocols for IP Flow Information Export(IPFIX)),以及更多的 RFC 草案。

2.1.5　流量测量的分类

网络流量测量从实时性角度考虑,可以分为实时测量和离线测量。

网络流量测量从测量方式考虑,可以分为主动测量和被动测量。主动测量(Active Measurement)主动生成网包并插入网络中,记录其传输过程的参数,例如,为了测量节点间的延时,向网络中发送 ping 包等。被动测量(Passive Measurement)被动观察网络中的网包,不生成新的网包。

网络流量测量从流量测量的环境考虑,又可以分为有线和无线环境的测量等;从实现的方式考虑,则有硬件实现和软件实现等;从测量的范围考虑,又可分为全部测量和部分测量(例如,只记录大流或者采样到的流)等。

在本章中,主要讨论的是有线环境中的实时流量测量,采用被动测量方式。

2.1.6　离线测量与实时测量

离线测量指离线分析存储下来的网包,离线测量模型如图 2.1 所示。

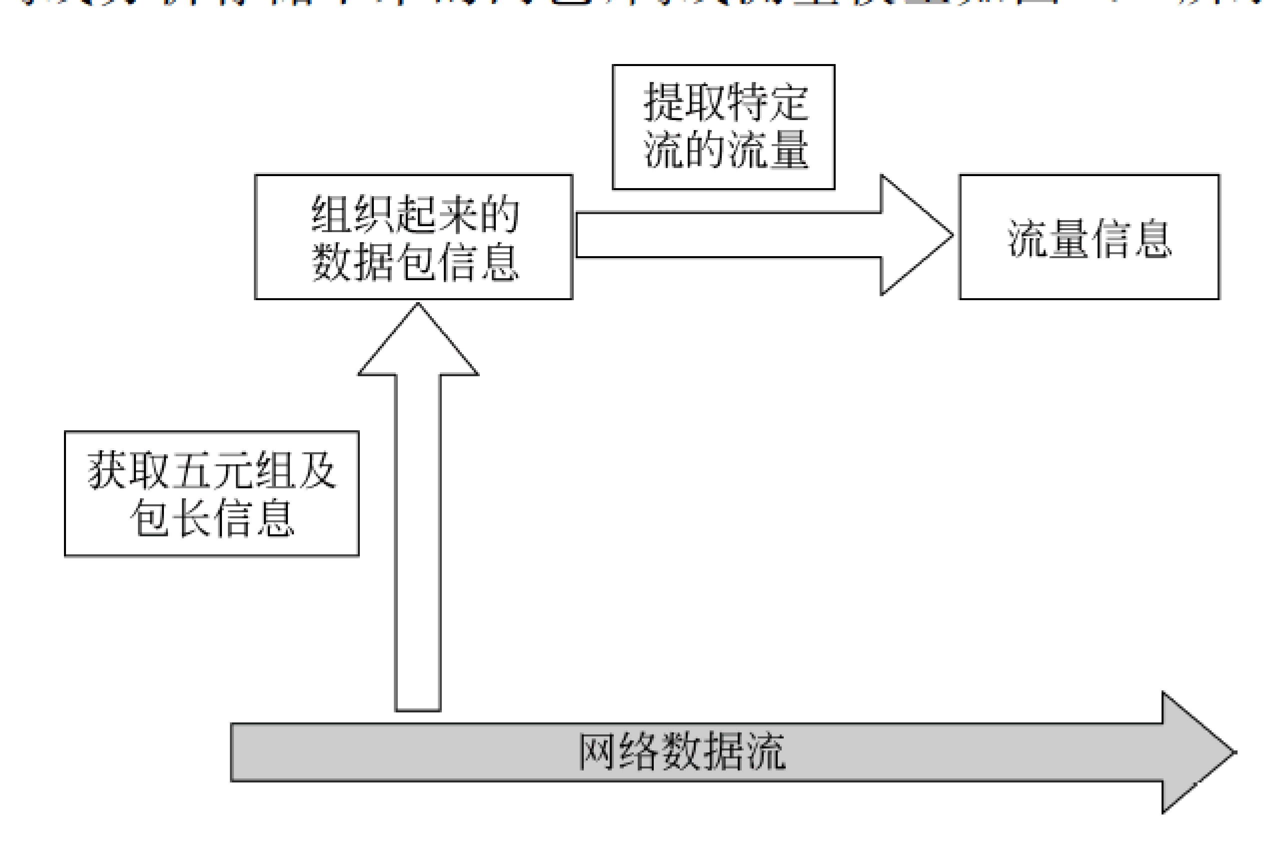


图 2.1　离线测量模型

实时测量指实时处理网络网包,实时测量模型如图 2.2 所示。



图 2.2　实时测量模型

离线测量的优点是不增加网包通过时的延迟，但在发现入侵时，异常网包已经进入子网。而实时测量则保证了每个网包都必须通过检查才能放行，能更好地控制网络流量。

在算法评估中，选择实时测量模型，因为它能充分发挥多核处理器的并行性，并且满足入侵检测和阻止的要求。

2.2 现有算法研究工作

2.2.1 原始记录算法及其存在的问题

原始算法就是最简单的流量测量的方法，为每个流保存一个计数器，每次接收到一个包后，遍历到相应的计数器并增加数值。为了统计流量的方便。选择 Hash 表的数据结构。如果出现 Hash 冲突，则用链表把冲突的元素连接起来。这种方法存在 3 个问题。

(1) 空间复杂度大。最简单的流记录由流标识和流量值组成。直接记录每个流的五元组需要 13B(源 IP、目的 IP 各需 4B，源端口、目的端口各需 2B，协议类型需 1B)，记录流量约需 5B(考虑一个 10Gbps 的网络一个小时可能的最大流量，以字节(B)为计数单位，需要长度为 $\log_2\left(\frac{3.6\times10^{12}}{8}\right)\approx39\text{b}$ 的计数器)，故共需 18B。因此当网络中一小时的流的数目达到 1.7×10^6 时，记录一个小时内每个流的流量，需要 144Mbit 的存储空间。一天的流量记录需要的空间在 Gb 以上。

(2) 时间复杂度大。由于为每一个流都保存了单独的计数器，接收到每一个包都需要更新计数器，因此处理包的时间复杂度不小。而在 10Gbps 以太网络中，如果要达到线速转发，最坏情况下(即处理包长 40B 的最小包)，对每个包的处理时间为纳秒级别：$T\leqslant\frac{40\times8\text{b}}{10\text{Gbps}}=32\text{ns}$。因此，即使是处理每个包的时间、空间复杂度随收到网包数目线性增长的算法都不能满足要求——当网包数目越来越大时，复杂度会增大到不可接受，例如，使用最大堆来寻找流量最大的 k 个流等(复杂度为 $2N+k\times\log(N)$)。只有 $O(1)$ 复杂度的算法可以被接受。

(3) 存储器的选择。在记录流量时，需要遍历内存以找到正确的流记录，修改流记录并写回内存，而内存访问的延迟是很大的。若使用 SRAM 存储，速度快，可以达到纳秒级别，但是价格贵，只能作为 Cache 或者少量数据存储使用；DRAM 价格便宜，可以大量使用，但是速度慢，影响系统吞吐率。这种状况加剧了前两个问题的困难程度。

2.2.2 解决问题的思路

解决上述困难有 3 个思路。

(1) 改进测量平台。例如，采用线程并行的方法减少内存访问的代价，即多线程并

行，当某一个线程开始内存操作时，进入挂起状态，直到内存操作完毕的信号到达。期间 CPU 上运行其他线程的计算，从而提高了 CPU 的利用率，实际上“隐藏”了访存的延迟。Intel 多核处理器的硬件多线程是一个典型例子。

(2) 不再处理每一个网包。方法之一是以一定比例采样，只处理采样到的网包。

(3) 不再为每个流保存一个记录。这样既节省了空间，又减少了访问内存的开销。例如，只记录流量超过一定阈值的流，这样做的实质是在流中统计出现频率高的元素，这是一个古老的算法问题。或者采用特殊的数据结构，减少内存，尤其是快速的 sram 需求。

2.2.3　采样算法

采样算法体现了思路 2 中的思想，典型代表是 InMon 公司开创的 sFlow4 技术，CISCO 的 netflow7，以及各种变形，例如，自适应采样等。采用 sFlow 的算法为代表评价和实现采样技术。

sFlow 的采样分为对网包采样和定期发送两个部分。对网包采样指每 n 个包中采样一个包，实现方法之一是在一次采样以后，生成一个随机数 skip，之后每接收到一个包，将 skip 减 1，当减到 0 时采样这个包，同时再随机生成一个 skip 值，继续循环，sFlow 采样示意图如图 2.3 所示。

要求 skip 序列满足最终的采样率 s（即采样到的包和总数之比）为给定值，另外还应该比较均匀，需要考虑网络流的特性，例如，突发性、长相关性等。这里随机数序列 skip 的生成有很多种方法，其中的一个替代方法就是设置一个阈值 t，生成一个随机数 x(x：1～m)，若小于或等于阈值则采样，大于阈值则不采样。这样采样率即为 t/m。

2.2.4　Multi-Stage Filter 算法

并行过滤器的技术体现了 2.2.2 节思路 3 的思想，是 Cristian Estan 和 George Varghese 提出来的，基本想法是：抓大放小（Focusing on the Elephants，Ignoring the Mice）。使用类似 Bloom Filter 的方法，判断并记录大流。算法的依据是 Internet 流量的 Zipf 分布：在骨干网上，9%的流占据了 99%的流量。因此，记录 9%的大流就足以反映网络流量的大致面貌，这对于检测主机扫描、端口扫描等已经足够，从而大大减少了计算量和空间需求。

算法判定大流（Elephants Traffic）的方法如下。

在 SRAM 中设置 n 个向量 V_i，每个 V_i 由 b 个计数器组成。对每个接收到的包的标识符 F，进行 n 次相互独立的 hash：$K_i = h_i(\mathrm{F})$，将流 F 的流量累加到 V_i 的第 K_i 位对应的计数器 $C_i(K_i)$。

若 $C_i(K_i)$ 的值超过阈值（例如，链路总流量的 0.1%），就说它通过了第 i 个过滤器。通过所有过滤器的流被认定为大流并记录入 DRAM。并行多级过滤器如图 2.4 所示。

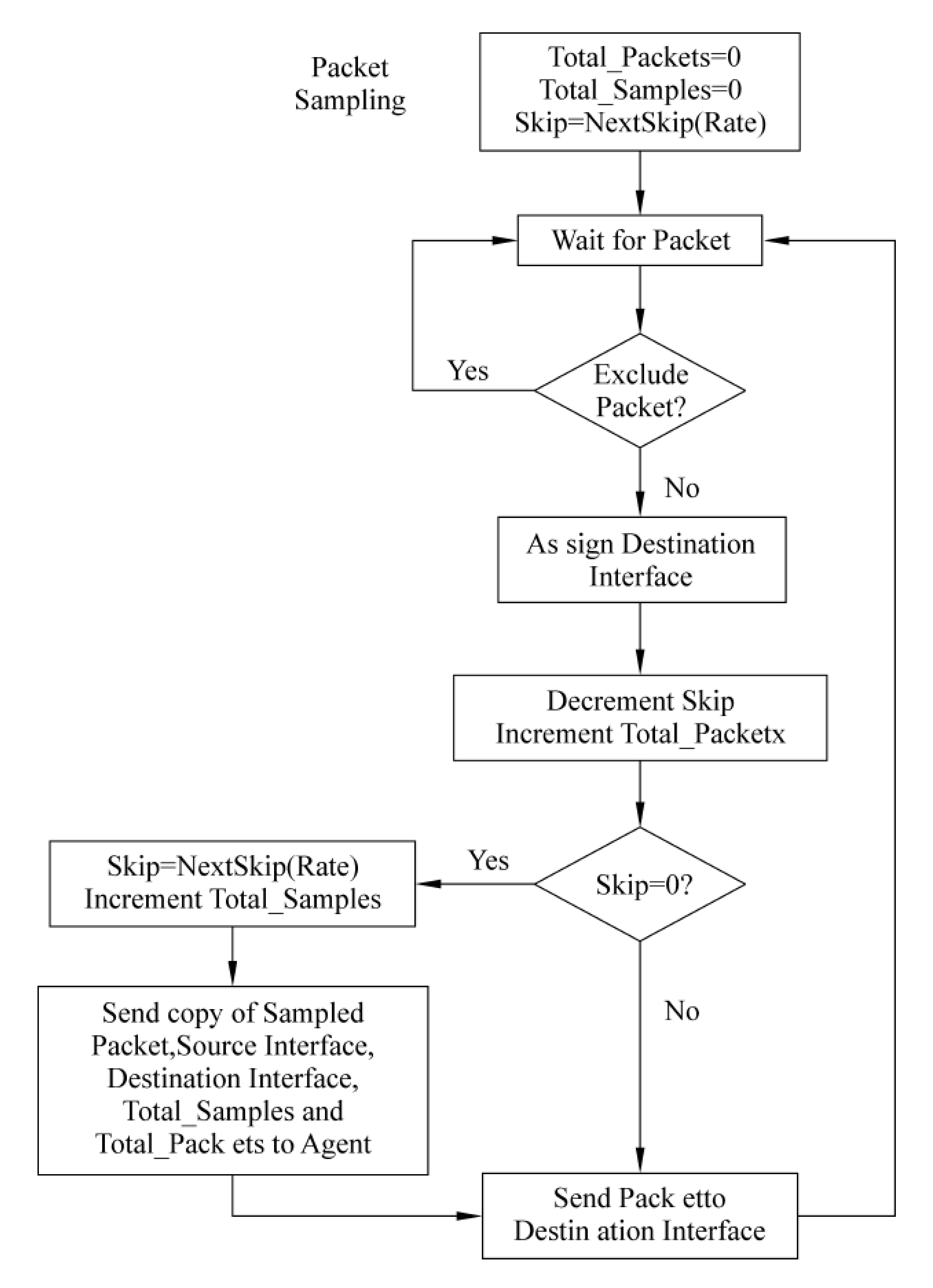


图 2.3 sFlow 采样示意图

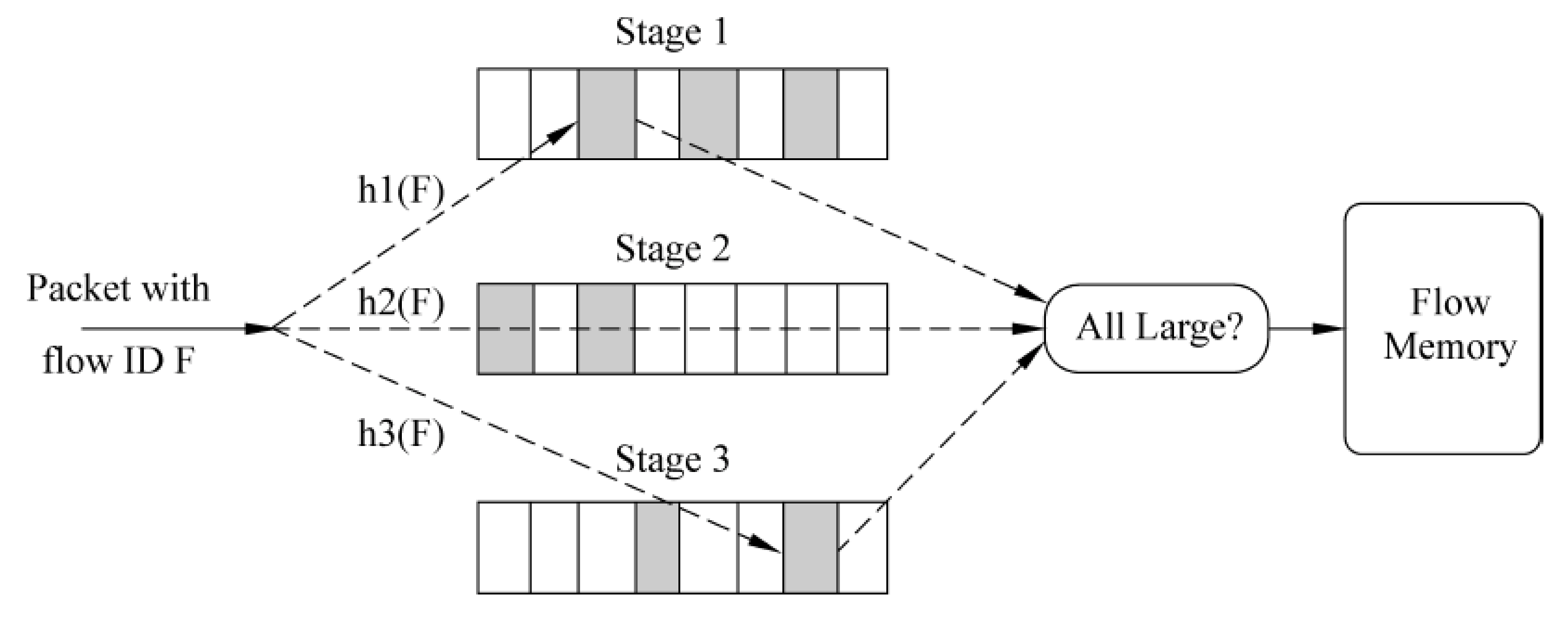


图 2.4 并行多级过滤器

Cristian Estan 等在同一篇论文里提出另一个类似的算法，只有那些通过了前面所有过滤器的流才会进入下一阶段的过滤器。串行多级过滤器如图 2.5 所示。

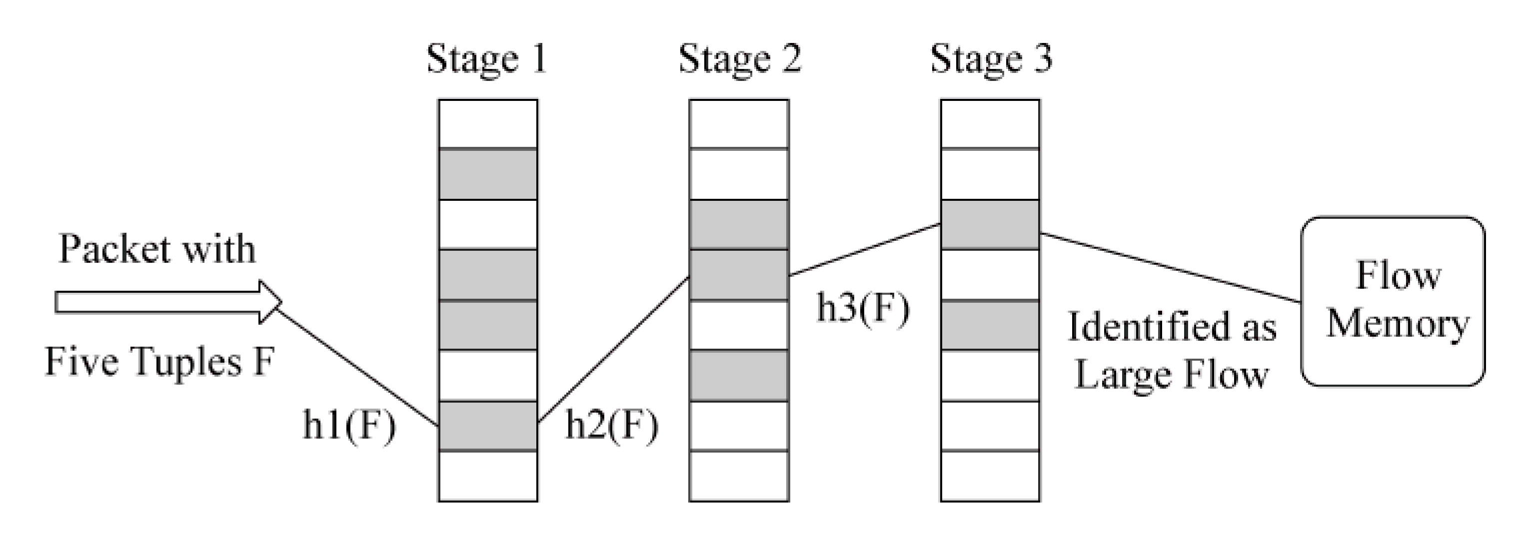


图 2.5　串行多级过滤器(Serial Multi-Stage Filter)

2.2.5　Multi-Resolution Space Code Bloom Filter 算法

多分辨率空间编码 Bloom 过滤器算法也体现了 2.2.4 节思路 3 中的思想。全称为 Multi-Resolution Space-Code Bloom Filter，是 Abhishek Kumar 等人提出的一种用于计数的数据结构。

先简单介绍用于字符串匹配的 Bloom Filter 算法。Bloom Filter 算法要解决的问题是：有一个字符串集$\{C_i\}(i=1\sim n)$，待查询的字符串 x，问 x 是否属于$\{C_i\}$。

设置一个长度为 m 的位向量 $\boldsymbol{BV}$，k 个相互独立的 hash 函数 $H_j(j=1\sim k)$，每个 hash 函数的取值范围是$[0,m-1]$。初始化时，把 hash 函数 H_j 作用于 C_i，得到 $H_j(C_i)$，将 $\boldsymbol{BV}$ 向量的第 $H_j(C_i)$位置 1。用 k 个 hash 函数把每个 C_i 都操作一遍，得到设置好的 $\boldsymbol{BV}$。

查询时，将 k 个 hash 函数作用于 x，得到 $H_j(x)$，$(j=1\sim k)$。查询 $\boldsymbol{BV}$ 上每个 $H_j(x)$位，如果都为 1，则 x 很可能属于$\{C_i\}$，否则，x 肯定不属于$\{C_i\}$。

当 hash 函数完全随机且无关的时候，如果 $p=\mathrm{e}^{-kn/m}$，则误判概率 f 为

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k\approx(1-\mathrm{e}^{-kn/m})^k=(1-p)^k$$

Bloom Filter 是一个时间和空间折中很好的匹配算法，Kumar 等将其改进成 Space Code Bloom Filter 用于流统计，其特点如下。

设置 L 个 Bloom Filter(BF)，对每一个流标识符 F，随机选择一个序号 $i(i=1\sim L)$，将 F 插入 BF_i 中(即用 BF_i 的 n 个 hash 函数作用于 F，将所得结果对应的 n 个 BV_i 的位置上 1)。

查询：若要查询某个流 F 出现了多少次，则查询每个 BF 是否包含 F，包含 F 的 BF 个数，即为流 F 出现次数。

但是当每个 BF 都较满的时候，即 $\boldsymbol{BV}$ 向量几乎都置 1，流的出现次数就被低估了。MRSCBF(Multi-Resolution SCBF，多分辨率 SCBF)解决了这个问题。

MRSCBF 的思路是：设置 r 个 SCBF，插入元素 F 的办法是以一定概率 $p_j(j=1\sim r)$将 F 插入每个 SCBF。概率 p_j 大的 SCBF“分辨率”高，概率小的“分辨率”低。当 $p_j=$

c_j-1 时，由于分辨率相互搭界，能很好地记录流 F。

查询：查询每个 SCBF 中 F 的出现次数，结合概率 p_j，用最大似然估计的方法估算 F 出现的次数。

2.2.6 其他算法

其他较有影响的算法还有 Bitmap 算法等，在此不详述。

2.2.7 工业界的解决方案

各大公司已经有很多的流量测量的产品上市，典型代表有 Cisco 公司的 Netflow、NetScout 公司的 nGenius 系统。

其中，Cisco 公司的 Netflow 已经成为实际上的标准。Netflow 的 IP 网包导出格式已经成为 RFC(例如，RFC3954：netflow version 9 等)。随着 IETF 对 IPFIX 的标准化，网络流量分析的数据采集协议也将逐步转移到 NetFlow v9/IPFIX 标准上来。其他公司如 Juniper 也基于 NetFlow v9 在路由器上实现了专有的流量测量模块。

2.2.8 NetFlow 介绍

NetFlow 技术最早是于 1996 年由 Cisco 公司的 Darren Kerr 和 Barry Bruins 发明的，并于同年 5 月注册为美国专利。NetFlow 技术首先被用于网络设备对数据交换进行加速，并可同步实现对高速转发的 IP 数据流(Flow)进行测量和统计。经过多年的技术演进，NetFlow 原来用于数据交换加速的功能已经逐步由网络设备中的专用 ASIC 芯片实现，而对流经网络设备的 IP 数据流进行测量和统计的功能也已更加成熟，并成为了当今互联网领域公认的最主要的 IP/MPLS 流量分析、统计和计费行业标准。NetFlow 技术能对 IP/MPLS 网络的通信流量进行详细的行为模式分析和计量，并提供网络运行的详细统计数据。

NetFlow 技术已经完全融入了 Cisco IOS 操作系统中。较新内核的 Cisco 公司的设备都内置有 NetFlow 功能，可以作为测量、采集、输出网络流量和流向管理信息的数据采集器。NetFlow 子卡实际上是为交换设备分担测量功能的加速卡，以达到减轻交换设备负担的目的。

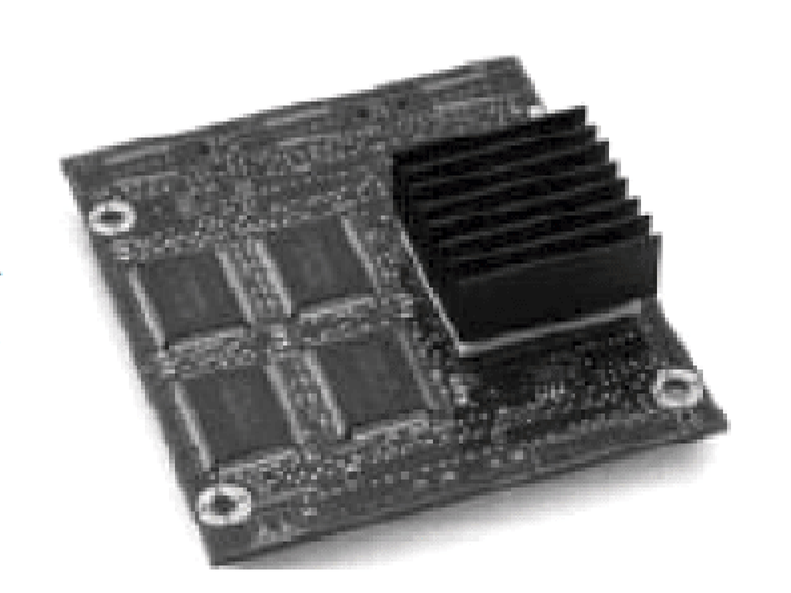


图 2.6 NetFlow 采集(服务)卡

图 2.6 是支持 Catalyst 4500 系列交换机的 NetFlow 采集(服务)卡示意图，是 Catalyst 4500 系列 Supervisor Engine IV 或 V 的一个可选子卡。

2.2.9 NetFlow 卡的工作原理

NetFlow 卡的多级处理流程如图 2.7 所示。

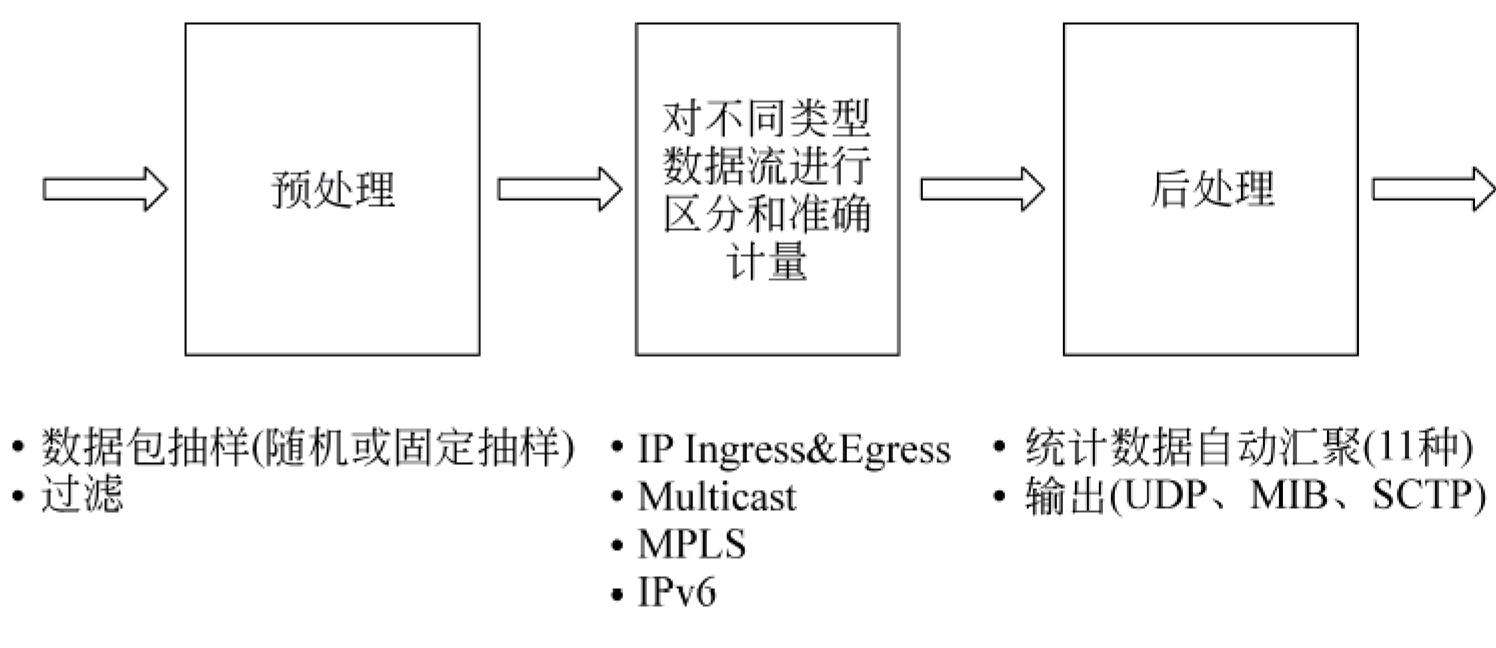


图 2.7　NetFlow 的处理流程

NetFlow 采集卡可以和 Cisco 公司的交换设备合作,直接获取硬件中的统计数据。在预处理阶段,NetFlow 可以首先根据网络管理的需要对特定级别的数据流进行过滤或对高速网络端口进行网包抽样,这样可以减少网络设备的处理负荷,增加全系统的可扩展性。

在后处理阶段,在新的 flow 不断生成的同时,Cache 内过期的 flow 记录以 UDP 网包等方式导出至某一预设 IP 的预设端口。NetFlow 可以选择把采集到的数据流原始统计信息全部输出;也可以选择由网络设备自身对原始统计信息进行多种形式的数据汇聚,只把汇总后的统计结果发送给上层管理服务器,从而大大减少网络设备输出的数据量,降低对上层管理服务器的处理能力要求。

NetFlow 根据 IP 网包的 7 个域,即源 IP 地址、目的 IP 地址、源通信端口号、目标通信端口号、第三层协议类型、TOS 字节(DSCP)、网络设备输入(或输出)的逻辑网络端口(ifIndex),来判断是否属于某一个流。

在 NetFlow 技术的演进过程中,Csico 公司一共开发出了 5 个主要的实用版本。

(1) NetFlow v1。为 NetFlow 技术的第一个实用版本。支持 IOS 11.1、IOS 11.2、IOS 11.3 和 IOS 12.0,但在如今的实际网络环境中已经不建议使用。

(2) NetFlow v5。增加了对数据流 BGP AS 信息的支持,是当前主要的实际应用版本。支持 IOS 11.1CA 和 IOS 12.0 及其后续 IOS 版本。

(3) NetFlow v7。CiscoCatalyst 交换机设备支持的一个 NetFlow 版本,需要利用交换机的 MLS 或 CEF 处理引擎。

(4) NetFlow v8。增加了网络设备对 NetFlow 统计数据进行自动汇聚的功能(共支持 11 种数据汇聚模式),可以大大降低对数据输出的带宽需求。支持 IOS12.0(3)T、IOS12.0(3)S、IOS12.1 及其后续 IOS 版本。

(5) NetFlow v9。一种全新的灵活和可扩展的 NetFlow 数据输出格式,采用了基于模板(Template)的统计数据输出。方便添加需要输出的数据域和支持多种 NetFlow 新功能,如 Multicase NetFlow、MPLS Aware Netflow、BGP Next Hop v9 和 Netflow for IPv6 等。支持 IOS12.0(24)S 和 IOS12.3T 及其后续 IOS 版本。在 2003 年 Cisco 公司的

NetFlow v9 还被 IETF 组织从 5 个候选方案中确定为 IPFIX(IP Flow Information Export)标准。

Netflow v5 的输出 UDP 包头格式如图 2.8 所示。

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
		Source ID
21	engine_id	Slot number of the flow-switching engine
22-23	reserved	Unused (zero) bytes

图 2.8 NetFlow v5 的输出 UDP 包头格式

在包头之后,可以跟 1～30 个详细的流记录。每条流记录长为 48B,详细格式如图 2.9 和图 2.10 所示。

Bytes	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	Interface index (ifindex) of input interface
14-15	output	Interface index (ifindex) of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes

图 2.9 NetFlow v5 导出数据中每个流的格式(0～36B)

37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol type (for example, TCP = 6; UDP = 17)
39	tos	IP type of service (ToS)
40-41	src_as	Autonomous system number of the source, either origin or peer
42-43	dst_as	Autonomous system number of the destination, either origin or peer
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (zero) bytes

图 2.10 NetFlow v5 导出数据中每个流的格式(37～47B)

2.3 流量管理概述

互联网从诞生至今已经历了将近半个世纪的发展,随着互联网的逐步发展,网上用户数量和业务流量也在不断增长,网络带宽正以每年一倍的速度增长,在骨干网络中已达到10/40Gbps 甚至 1Tbps 的规模。除传统数据业务外,网络电话、网络视频、P2P 文件共享等新型网络应用使得骨干网络中语音、视频、点到点传输流量都呈现几何级增长趋势,占据了大量的互联网带宽。然而,网络应用繁荣的同时,网络管理的难度也随之增加。传统的网络设备由于无法识别这些新生的应用层协议信息,致使应用级的管控无法跟上管理的需要,网络管理的灰色地带不断增加。同时,网络资源滥用严重,难以进行有效控制管理,一些关键应用的信息传输的低延迟和低抖动要求,更是难以得到有效保障,网络中的关键应用遭受严重影响。而且,由于网络的管理技术缺失,网络应用对带宽的需求增加,以及 DDoS 攻击等消耗网络资源为目的的流量类攻击发展迅猛,能够迅速造成网络拥塞,甚至网络设备瘫痪。对流量缺乏有效的识别与控制手段,使得网络安全性大大降低。

对于今天的网络管理者而言,如何深度智能感知网络应用,通过合适的带宽管理技术来解决带宽增长与业务收益、网络扩容与用户体验之间的不对称关系,实现对用户和业务的分级化识别管理,以及基于用户和用户业务流量的管理和增值显得尤为重要,学术界和工业界也都对此十分关注。为了有效监管与分析网络带宽的使用情况,防止视频直播、P2P 文件共享等应用过度占用网络带宽,维护良好的网络环境,亟须一种高速的网络流量管理系统,针对出现问题的时间、应用种类做优先级控制和优化。

2.3.1 流量管理定义

流量管理技术主要包括协议识别和流量控制。流量管理是通过对应用层协议的识别,来实现对不同的应用层协议实施不同的控制、限速和 QoS 服务。如何准确地识别应用层流量,并对各种流量实现合理控制是目前研究的难点。

1. 协议识别

协议识别是进行流量控制的前提,只有在精确识别应用层流量的基础上,对其进行控制才有实际意义。而对于应用层协议的识别,主要有基于端口、深度检测和基于行为的 3 种方法。

1)基于端口的应用层协议识别

使用端口来判断协议类型是最早使用也是使用最为广泛的识别方法。Internet 端口分配机构(Internet Assigned Numbers Authority,IANA)负责对 IP 地址分配规划以及对 TCP/UDP 公共服务的端口定义,它规定了常用协议的默认监听端口,比如 HTTP 的端口是 80,POP3 的端口是 110,SMTP 的端口是 25。根据端口对常用协议进行识别判定简单、快速,这也是端口判定法至今还被应用在高速网络处理中的主要原因。

然而网络高速发展,各种协议层出不穷,大量的应用层协议为了避免识别、逃避防火墙的检查,不使用固定的端口进行通信。单纯依据端口来判断协议已经无法达到准确性方面的要求。这不仅包括众多近年新出现的 P2P 协议,而且包括了越来越多的传统协议。传统的基于常用端口的应用层协议识别方法已经不能适应于当前的互联网网络管理的需求。

为了弥补根据端口识别协议无法解决的问题,近年来很多的研究工作都致力于开发新的方法来识别应用层协议。而将基于端口识别的方法与其他方法相结合,也是非常不错的选择。

2)基于深度检测的识别算法

基于深度包检测(Deep Packet Inspection,DPI)的识别算法首先通过分析各种应用层协议,提取出一些能对具体协议进行表示的载荷部分(Payload)的特征(Signature),组成一个特征库。对网络流量进行识别时,通过将网包的载荷部分与特征库中的特征进行匹配以识别出各种应用层协议。这种识别算法不仅能识别出使用单一连接进行通信的协议,而且能够识别出如 PASV FTP、流媒体等使用多个连接、动态端口进行通信的协议,是目前运用最普遍的应用层协议识别方法。但是,这种方法还存在一些不足,比如对于加密传输的应用层协议,将载荷与特征字符串进行匹配的方法就几乎无法检测。

总的来说,这种方法的准确性是目前所有算法中最高的,误报率一般小于 10%。但由于需要逐网包的匹配所有协议的特征以及额外的存储网包的负载部分,该类算法的时

间和空间复杂度很高，并且随着待识别协议数量的增长而迅速增长。所以在不降低识别准确度的基础上，如何提高算法性能是研究的主要内容。

3）基于行为模式的识别算法

基于行为识别协议的算法利用协议规范的不同所造成的流测度的差异区别各个协议。例如，Web 流一般为短流小网包，而 P2P 流一般为长流大网包。基于行为模式的算法要求事先有标准的训练集，即要用已按各个协议分类的报文集合来训练识别器，使其在使用的过程中根据已知的标准答案和新计算的流测度，按照某种判别算法得出当前流所属的类别和使用的协议。这种算法虽然在精度上目前无法达到基于特征的识别方法，但是由于它具有强大的处理加密流量和未知流量的能力，因此也是协议识别方法的一个重要研究方向。

从理论上说，每种协议由于其规范不同，行为模式总会有所不同，因此，基于行为的算法理论上也可以识别所有的协议。但是，当前所研究的算法都只能将流分成几大类（例如，bulk transfer、single and multiple transaction 和 interactive traffic 等），没有达到识别协议的最终目的。此外这一算法的准确率一般都不高（低于 80%），研究尚不成熟。

2.3.2　流量控制

当前的网络容量不能满足日益丰富的网络应用的需要，更多、更合理的控制机制对已有网络的稳定运行无疑是至关重要的。一个完整的流量控制结构必然要在恰当的层次和粒度上对流量进行必要的管理，其中包括流量整形、队列管理和网包调度等。

1. 流量整形

流量整形是调整网包传输的平均速率，使网包按照传输模式规定的速率进行传输，尽量避免突发性流量导致的拥塞问题。主要方法是控制队列的输出，以调整输入网包的速率和平滑传输速率。产生突发流量的原因是因为网络传输是以网包为单位的，传输两端一般是产生突发的网包序列，而且网络中间设备很难对突发进行预测。例如，一个用户接入网的平均带宽为 X Mbps。X Mbps 就有多种情况，每秒传输 X Mb 的网包是最为理想的情况；然而用户流量可能在前 1s 发送 $2X$ Mb，然后停止 1s。在这种情况下，用户的平均带宽仍然是 X Mbps，但是突发带宽就是 $2X$ Mbps。如果多个用户同时发送突发网包，虽然平均带宽符合设置，但是在突发瞬时网络已经变得拥塞，甚至造成网包丢弃。

2. 队列管理

队列管理的作用是在网包到达队列前端时依据一定的策略和信息决定是否允许该网包进入缓冲队列，队列管理是流量控制的重要手段之一。列缓冲队列的设置提高了带宽利用率和系统吞吐率，然而同时也增大了网包排队延迟，在队列缓冲空间的容量增大时更为严重。因此，如何设置合适的队列缓存容量，如何在网络运行期间合理地控制队列长度

的动态变化，以平衡系统吞吐量和网包排队延时之间的矛盾，是队列管理的首要问题。典型的队列管理算法有自适应虚拟队列（Adaptive Virtual Queuing，AVQ）、成比例丢失率控制算法（Proportional Loss Ratio，PLR）和随机早期预测（Radom Early Detect，RED）。

3. 网包调度

基于多队列的控制方案在业务隔离、资源公平分配等方面是实现区分服务的必然选择。网包调度是指按照一定的规则来决定从哪一个等待队列中选择哪一个网包进行发送，使得所有输入业务能够按照预定的方式共享输出的链路带宽。网包调度规定了网包从每个队列离开的瞬时特性，通过流隔离，使得不同的业务的网包获得不同等级的服务。根据工作原理和控制目标，调度算法可以分为以下几类：基于优先级、基于轮询、基于GPS (Generalized Processor Sharing)模型、基于时延和分层链路共享算法等。

如果一个系统中同时实现以上 3 种算法，虽然可能达到比较好的控制效果，但是系统处理的开销大幅上升，时延增大。所以在实际应用的系统中一般会根据需要选择实现一到两种算法。本文的研究将集中于流量的带宽分配和流量整形，其余功能也能够类似扩展。具体的网包调度算法将在 2.3.3 节给出详细介绍。

2.3.3 高速流量管理调度算法比较与分析

1. 应用背景

综合服务 IntServ（Integrated Service）和 DiffServ（Differentiated Service）是两种不同的 Internet QoS 体系结构。

IntServ 同时针对单播和多播，由接收方发送资源预留消息，沿着生成树上溯到发送方；在沿途的每一跳，路由器看到此预留消息，并保留必要的带宽；当预留消息抵达多播源时，由发送方到接收方已经预留了带宽。该体系结构主要用到资源预留协议（Resource reSerVation Protocol，RSVP），提供一种基于流的细粒度的服务。然而当流的数目非常之多超出带宽可承载能力的时候，这种做法的扩展性非常差；同时路由器内部需要为每个流维护一份内部状态，这使得路由器易崩溃；此外，路由器之间需要进行复杂的消息交换。鉴于以上三点，基于 RSVP 的 IntServ 很少应用于实际。

DiffServ 则是将服务按服务质量（Quality of Service，QoS）分类，给每个类别分配一定的资源，每个服务类别与相应的转发规则相对应，服务类别较高的服务得到的服务具有较大的带宽，较低的延时，较低的抖动，较小的丢包率；同时可能需要支付较高的费用等。

这里的调度算法皆基于 DiffServ 体系结构，也就是当网包到达路由器后，按服务要求、服务类别分到不同的队列，该如何在队列之间、队列内部进行网包传输的先后顺序的决策，以尽可能满足各个服务类别在 QoS 上的要求，将是网包调度算法所考虑的问题。当然，其中有部分算法也可用于 IntServ。

至于调度算法在系统中的存储位置，既可以在路由器的输入接口处进行，也可以在路由器的输出接口处进行，也可以结合起来，在输入输出接口处同时排队。

2. 网包调度综述

1）问题提出

在高速网络中，网包到达路由器等网络中间设备，需要进行网包分类、路由查找、队列管理等动作；而由于网包大量到达，需要在路由器向链路输出时决定网包的输出顺序，因而还需要进行网包调度，如图 2.11 所示。

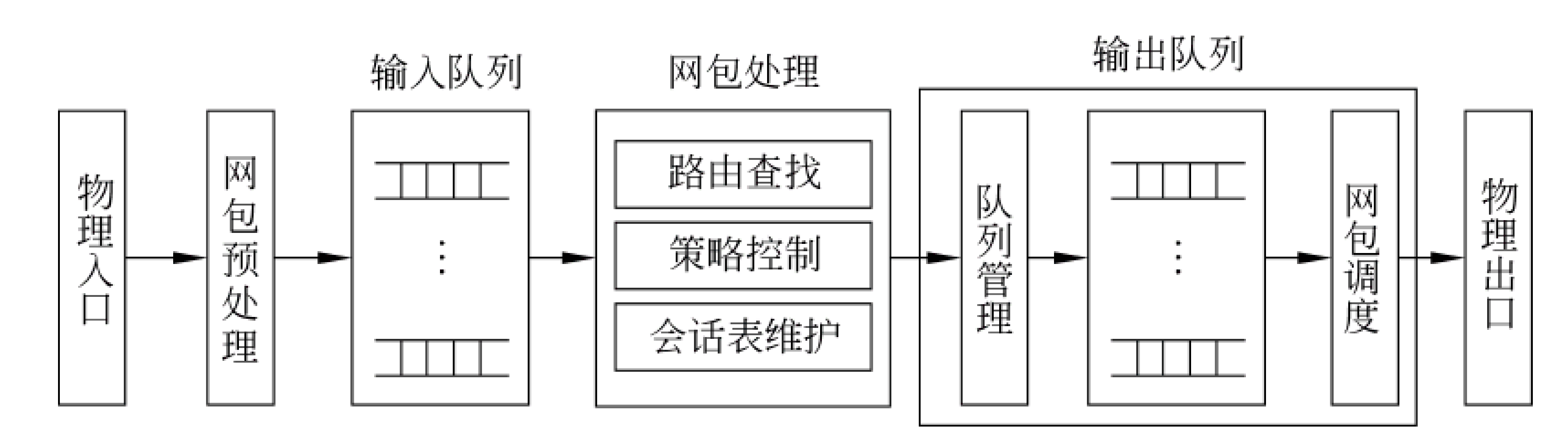


图 2.11　网包调度在路由器中的位置

当网包大量到达路由器等网络中间设备时，容易导致数据在网络中突发、堆积，此外还可能无法满足部分实时数据的传输延迟限制，因此需要设计一种策略，既限制突发流量引发的拥塞，调整流量趋于平滑，又满足不同用户的数据传输策略。因此想到将网包按照一定的特征分到不同的队列中，按队列特性和网包的时间特性决定网包的先后传输顺序——这就是调度过程。而决定传输顺序的策略，就是调度算法。

调度算法本质上是要解决带宽资源争用的问题，具体来说有如下几方面的问题。

（1）公平的分享带宽。按照带宽提供量合理满足不同队列对带宽的需求，同时不顾此失彼。

（2）对每个用户提供最大最小的带宽保证。实际上保证为某用户的队列所提供的服务量在单位时间内不能超过最大带宽，不能小于最小带宽。

（3）保证丢包率的限制。在某些情况下，队列有一定的容量，队长超过该容量后，可能丢弃某些包；分配给队列的带宽越大，丢弃率可以越小。

（4）保证延时的限制。

（5）保证延时抖动的限制。保证队列流量基本平滑，尽量不出现某个包延时大、某个包延时小的情况。

（6）保证要求的调用控制、策略、整形、丢弃策略、缓存分配和调度。

（7）不同队列间隔离。某个队列流量的突发不会影响到其他队列满足服务要求。

2）衡量算法优劣的几个指标

（1）调度算法的时间复杂度。也就是调度算法确定下一个网包需要花费的时间。

(2) 公平性。公平性有多种定义，这里以尽量满足用户需求，合理分配系统资源为公平。指标有 B-WFI、T-WFI 等。其中带宽最坏公平指数 B-WFI (Band Worst-case Fairness Index)用来表示队列在实际网包系统和相应理想流系统上接收到的服务量的最大差值(见公平调度算法类章节)。SFI(Service Fairness Index)用来比较在同一时段内，任意两个流所接受的服务量的差值(具体解释见 4.1 节)。

(3) 延时界。也就是在最坏情况下的延时和延时抖动需要一定的范围，与其他流的行为、流的数目、其他流的预留资源无关。

(4) 算法的实现是否简单有效。

3) 定义调度算法

现有的调度算法有一些分类方式，比如一种区分方式是持续工作(Conserving-work)和非持续工作(Non-conserving-work)。持续工作就是有任务时，服务器不会空转，这可以保证低的平均延时，高总吞吐量，但也可以导致高延时抖动。非持续工作指即使有网包在队列中等待，也可能先不要调度，而是需要先进行流量整形，这可以保证在最坏情况下，端到端延时更小，产生更高的网络吞吐量，而且还能保证低的延时抖动；但每一跳都整形可能会导致额外的跳延时。实际上，各种算法的关系错综复杂，无法严格地按某种机理区分它们。在研究已有算法的过程中，发现有些算法以时延作为调度依据，有些算法以带宽作为调度依据，另一些则以速率作为调度依据(也就是综合考虑了带宽因素和时延因素)。有些算法只考虑单个调度器对业务流的处理，另一些算法则考虑包分类、缓存管理等之前之后的因素，乃至整个网络的排兵布阵。

通过对这些调度算法的研究，定义调度算法如下：在高速网络中，对于共享同一链路、不同种类的业务流，考虑其时延、带宽等 QoS 要求，需要在尽可能满足每条业务流的 QoS 需求的情况下给出网包接受服务的先后顺序的一种算法。

不同的网络环境、不同的应用程序对性能的要求不同，因而对调度算法的要求也不同，比如有些网络需要优先考虑时延，有些网络需要优先考虑公平性，有些网络需要为某些特定流保证绝对的实时性等。

4) 调度算法思想

调度算法本质上是将业务流分类，作为不同的队列，再给每类业务流一个权重，一个以时间为参数的限制曲线，该曲线与权重有关；算法设计要求队列间的选择满足该队列的曲线限制，队列内一般采取先进先出的做法。权重有多种形式，比如等权重、优先级、速率比和带宽限制；曲线也有多种形式，比如速率上下限和带宽上下限，因而调度算法有多种形式。

各种纷繁复杂的算法皆基于一些基本的调度思想，由这些思想演变而来。这里罗列其中几种。这些思想的实现本身也可以作为调度算法。

(1) 先进先出(First In First Out，FIFO)的调度思想。

先进先出作为算法，各种网包不分类，也就是只有一条队列，按到达顺序进入该队列中按次序接受服务。在这种情况下，首先不能保证很好的公平性，其次用户间不隔离。很多高级的调度算法将网包按某些方式分类，每一类一个队列，队列间的调度有许多可做的工作，而队列内部通常采用先进先出的做法。

（2）基于优先级队列（Priority Queuing，PQ）的调度思想。

给每个队列划分优先级 0，1，2，…，$n-1$，优先级数字越小，优先级越高。从优先级高的队列开始调度；当且仅当优先级 0，1，…，$i-1$ 的队列为空时，优先级为 i 的队列才可以被服务。

优先级较高（数字较小）的队列，能够保证较小的延时，较大的吞吐量，较小的丢包率。如果优先级高的队列过载（需要传输的网包过多），优先级较低（数字较大）的队列容易被饿死（很长一段时间内得不到服务）。因此这类算法公平性很差。

（3）停等（Stop and Go）的调度思想。

将时间分为时间片是该算法的关键。在空闲时间片伊始，轮询各个队列，一旦队列有网包，则传输网包，直至传输完毕；在下一时间片伊始，从下一队列开始，再次轮询队列。

QSP 结构如图 2.12 所示。

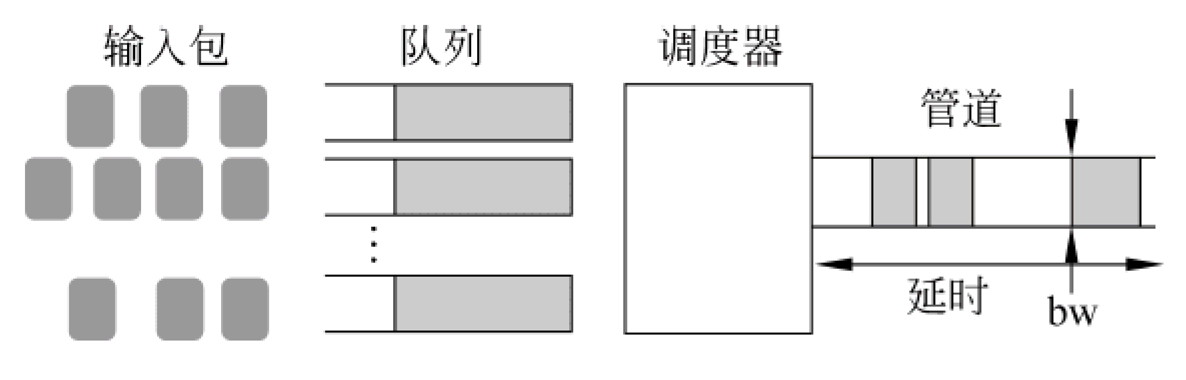


图 2.12　QSP 结构

5）调度算法的 QSP 结构

本节所述涉及包分类队列的调度算法皆基于一种多队列—调度器—管道 QSP（Queue-Scheduler-Pipe）结构。该结构分为前端的队列，中间的调度器，后端的传输管道。输入包按一定的规则分发到各个队列中，调度器按一定的调度算法从各队列中取出网包送入管道进行传输。

6）基本概念

为了方便读者，这里罗列并解释一些关键术语，这些术语频频出现在各种文献中。

（1）包。在这里是网包调度中网包的意思，也就是网络层的数据单元，因为部分算法中有队列网包，为了避免两者混淆，统一称为网包。

（2）服务量。在网包调度中即链路传输量。

（3）流、队列、业务流、queue 几个概念等价。

（4）传输链路。反映到上述结构中，即 pipe（管道）。

（5）L_{max}。最大包长。一般是 MTU（Maximum Transmission Unit）。

3. 基于轮循的算法类

各种 RR(Round-Robin)调度算法的核心是将时间片分给队列的分配方式。一旦某个队列获得时间片,将发送一定数目的网包到输出链路。

1) 轮循调度 RR

对到达的包进行分类,分到各个队列中;在队列间轮循调度,每次调度当前队列的一个网包进行传输,传输结束后至下一队列;如果当前队列为空,则直接询问下一队列,以此类推。然而基本轮询算法忽略了包的长度和时间需求,不是公平算法。

2) 带窗口的基于优先级的调度

该算法本质上是一种轮询算法。为优先级为 i 的队列指定一个最大服务包数 n_i(窗口大小),对所有队列按照优先级的高低进行轮询,每轮从优先级最高的队列开始,优先级为 i 的队列每轮可以服务至多 n_i 个网包;只有 n_i 个网包被服务之后,才会询问更低优先级的队列。这样可以保证具有低优先级的队列中的网包不被饿死,但这样一个轮询的过程会增加高优先级队列的延时。

特别地,当 n_i 比较小时,该调度算法趋近于 RR 算法;当 n_i 比较大时,每一轮在优先级较高的队列中服务比较久的时间,该调度算法趋近于 PQ 算法。

n_i 和延时或丢包率之间的数量关系有待研究。具体参阅 El-Gebaly 和 Sabaa 等人的工作。

3) WRR(Weighted Round-Robin)

给每个队列赋予不同的权值,代表一次完整循环队列被服务的网包数。同时为每个队列维护一个计数器,初始化为权值。每次轮循时,计数器为非零的队列允许发送一个网包,同时计数器减一。当所有队列的计数器为零时,重置各队列的计数器为各自权值。加权轮循算法考虑不同队列对服务量的需求,但未考虑包长因素,也不是公平算法。

4) DRR(Deficit Round-Robin)

在 WRR 的基础上考虑包长因素,以字节为单位为每个队列分配一个带宽份额,同时为每个队列维护一个计数器,初始化为权值。每次轮循时,如果待发送包的包长小于计数器的值,则发送该包,同时将计数器的值减去包长值。如果待发送包的包长大于计数器的值,则继续询问下一队列,同时当前计数器的值在下一轮轮循至该队列时,将此剩余值与权值之和分配给该队列作为计数器(亦即每轮轮询结束后,为每个队列重新分配带宽份额,并加上原来的剩余值)。DRR 很好地解决了带宽分配的公平性问题,而且具有 RR 算法的共同优点:制定决策的时间是 $O(1)$ 的。但在时延需求方面,在不考虑权重的情况下,需要等待其他队列被服务一轮之后,才可接受下一次的服务。另一方面,由于每一轮轮循都会给每个队列分配权值大小的带宽份额以累积计数器的值,一旦某队列轮空多轮后,突发大量流量,这些流量都可能在一个轮循周期内传输,这就造成流量突发。许多高

速路由器的调度算法(如 Cisco GSR 等)都是基于 DRR 原理。

另有一种 Aliquem 算法是对 DRR 的改进,改进之处在于,每轮分配给队列的份额按比例缩减,这样可以降低突发性。

5) SmRR(Smoothed Round-Robin)

该算法是 DRR 的改进,主要解决流量突发的问题。对 DRR 的改进在于,每轮分配给队列的份额值不是一次增加很多,而是每次只增加一个最大包长 $L_{\max}$,这种情况下,即使有剩余份额的积累,也不会产生太大的突发流。

为了说明 SmRR,先解释两个概念——WSS 和 WM。

WSS(Weight Spread Sequences)递归定义如下:

$\mathrm{WSS}_1=1$

$\mathrm{WSS}_n=\mathrm{WSS}_{n-1},n,\mathrm{WSS}_{n-1}$

举例如下:

$\mathrm{WSS}_1=1$

$\mathrm{WSS}_2=1,2,1$

$\mathrm{WSS}_3=1,2,1,3,1,2,1$

$\mathrm{WSS}_4=1,2,1,3,1,2,1,4,1,2,1,3,1,2,1$

$\mathrm{WSS}_5=1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,1,3,1,2,1,4,1,2,1,3,1,2,1$

很容易看出来,理想情况下,在 WSS_n 序列中,j 出现的次数是 $j+1$ 的两倍。扫描 WSS_n 序列,扫描到 j 表示调度权值含 2^{n-j} 项的队列。那么可以看出权值含 2^{n-j} 项的队列被调度的次数是含 $2^{n-(j+1)}$ 项的队列被调度的次数的两倍。当然,这是一个极限情况,或者队列被扫描到时,有网包可发送的情况。

WM(Weight Metrix):该算法给每个队列分配一个权值,该权值表示成 2 的幂次和的形式,比如对队列 i 分配的权值为 $w_i=a_{i,n-1}\times 2^{n-1}+a_{i,n-2}\times 2^{n-2}+\cdots+a_{i,0}\times 2^0$,$a_j^i=0$ 或 1,如此一来,有网包要发送的队列 $1,2,\cdots,N$ 足以生成权值矩阵

$$\mathrm{WM}=\begin{bmatrix} a_{1,n-1} & a_{1,n-2} & \cdots & a_{1,0} \\ a_{2,n-1} & a_{2,n-2} & \cdots & a_{2,0} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,n-1} & a_{N,n-2} & \cdots & a_{N,0} \end{bmatrix}$$

WM 从左至右每一列依次标记为 $1,2,\cdots,n$,按照序列 WSS_n 指示为 WM 的列,按列调度。

具体来说,调度过程如下:扫描一遍 WSS_n 为一次轮询,当扫描到 WSS 的指示为 j 时,选取 WM 第 j 列中所有非零项,对应到 WM 的第 i 行,为队列 i 的份额增加一个 $L_{\max}$,按照份额服务于队列 i;当该列所有非零元素对应的队列都被服务一遍后,扫描序列 WSS_n 的下一指示。

WM 矩阵的形成：由于 WM 矩阵中的每一行标定了一条激活流，那么存在一个添加激活流和删除非激活流的过程。具体来说，当有新的网包到达而且需要分配为新的数据流时，生成一条激活流。为这条流计算权值的二进制幂次和，并添加作为 WM 新的一行。那么一旦这条流的权值有不低于 2^n 的项时，需要在 WM 中添加列。这时对应 WM 矩阵中其他激活流前几列添 0。

SmRR 是在 DRR 基础上对权值做了一定的处理的一种很精巧的设计，形式上很有些杨辉三角的味道。

6）RRR（Recursive Round-Robin）

RRR 依赖的数据结构是一棵加权二叉树（Weighted Binary Tree，WBT），树的第 i 层中的节点，表示权值为 2^{-i}。

该算法中有一个建立树和选择数据流进行传输服务的过程。对队列 i 分配的权值为 $w_i = a_{i,n-1} \times 2^{-(n-1)} + a_{i,n-2} \times 2^{-(n-2)} + \cdots + a_{i,0} \times 2^0$，$a_j^i = 0$ 或 1，然后将各个有网包需要传输的流按照顺序插入到树中，成为树的叶节点。队列 i 的权值中 $a_{i,j}$ 项为 1，意味着 w_i 将出现在第 j 层。当所有的流都插入完毕，便构造成为一棵 WBT。调度时，每次从根节点开始，一次向左子节点走，一次向右子节点走下去；当走到子节点时，同样严格遵循一次向左走，一次向右走；直到遇到叶节点，访问该节点，传输其所代表的数据流，如图 2.13 所示。

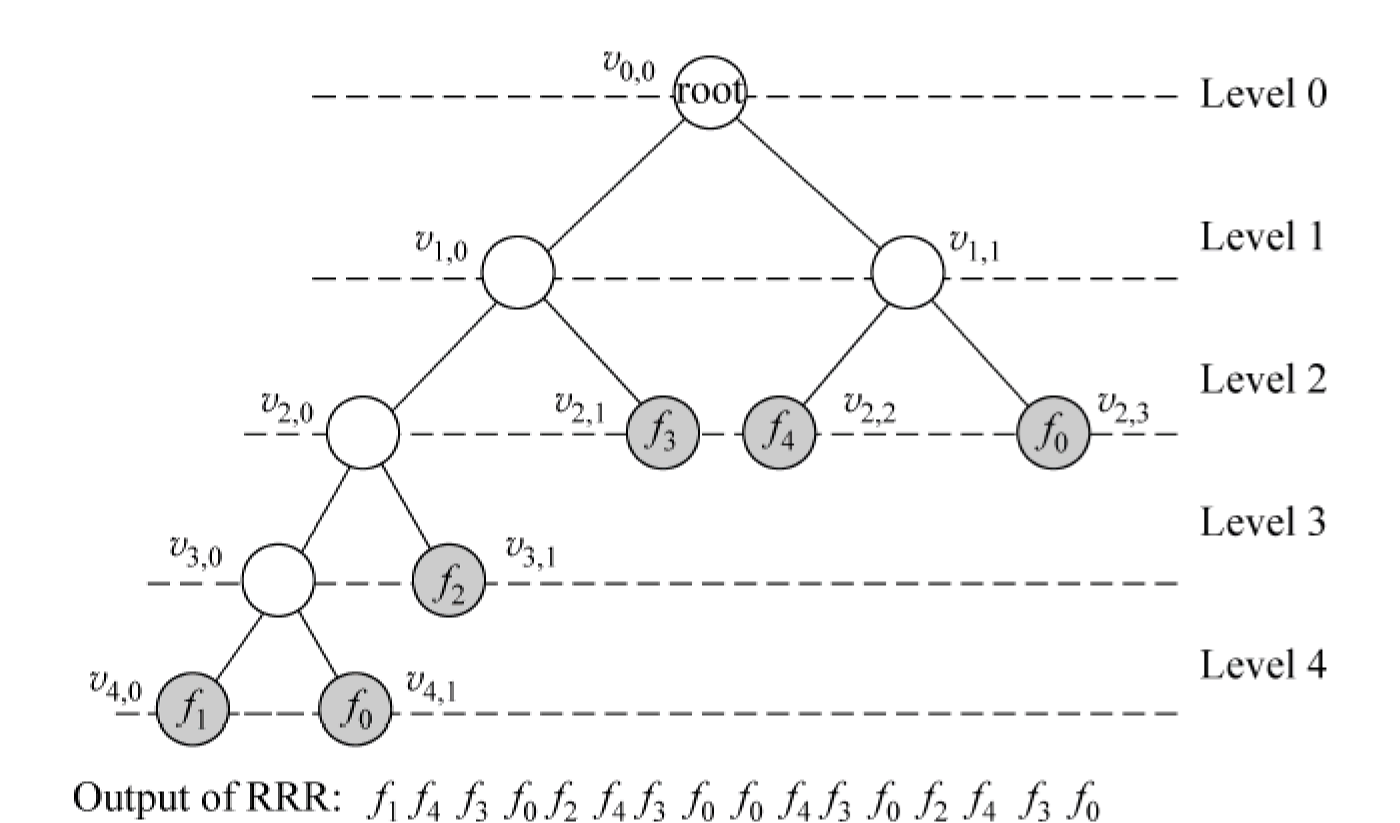
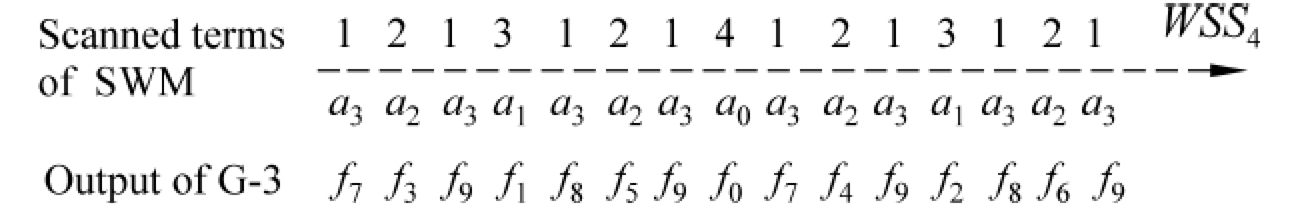


图 2.13　加权二叉树

该算法的关键之处是严格地按一次向左走，一次向右走，这样保证了流在 WBT 中层数字越小，被访问的次数越多；在理想状态下，第 i 层和第 j 层的业务流被访问的次数之比是 $\frac{2^j}{2^i}$。

7）G3

该算法结合了 SmRR 和 RRR，借鉴了 SmRR 的 WSS 序列，借鉴了 RRR 的 WBT，将其扩展为完全加权二叉树(Perfect Weighted Binary Tree，PWBT)，如图 2.14 所示。



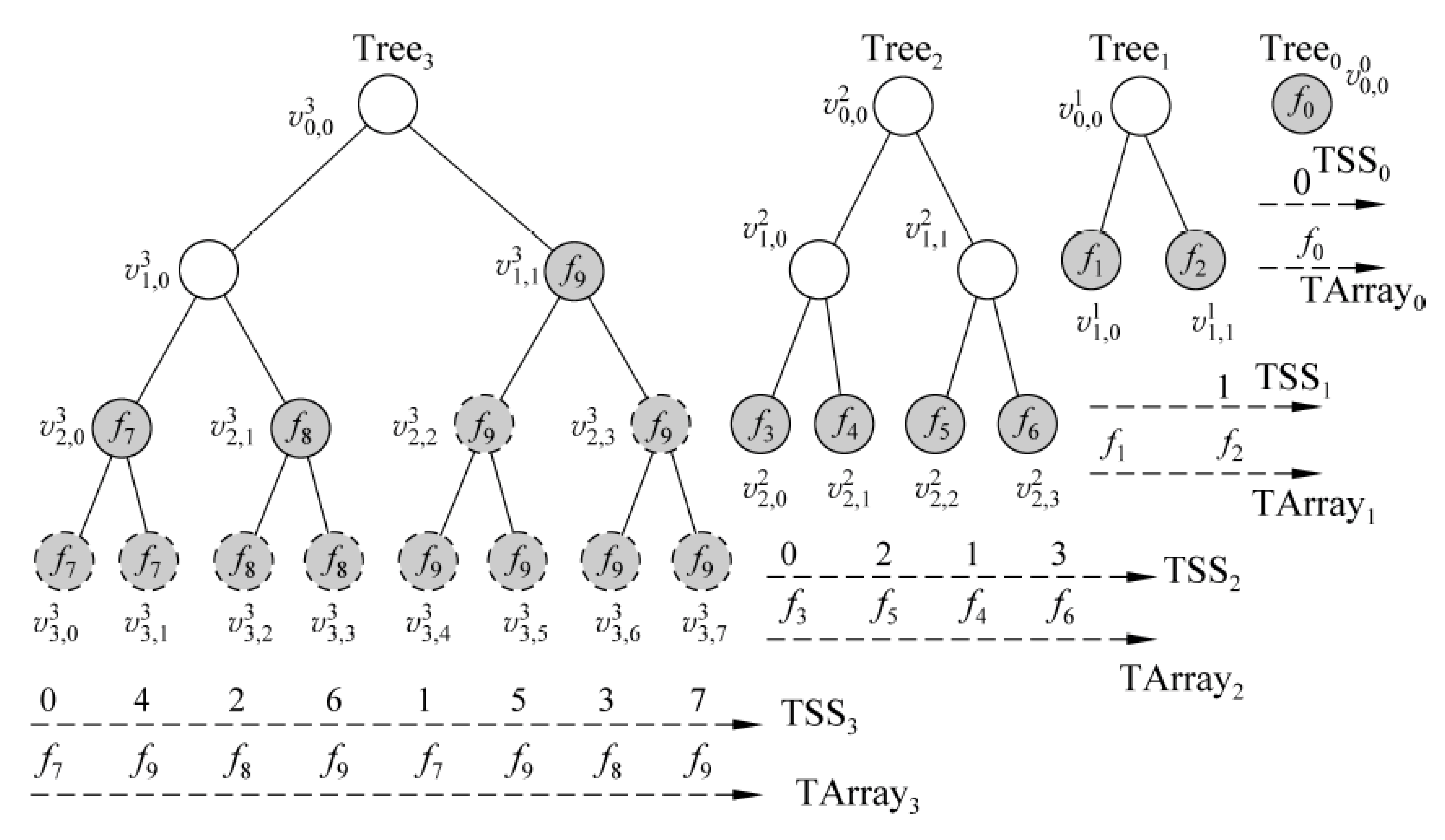


图 2.14　完全加权二叉树 PWBT

具体做法如下。

(1) 按照权值份额，生成一棵一层的 PWBT 树 Tree_1，一棵两层的 PWBT 树 Tree_2，一棵三层的 PWBT 树 Tree_3，……，一棵 n 层的 PWBT 树 Tree_n，……。将节点按所占份额从小到大排序，依次插入到这些树中。

(2) 按照 WSS 的指示访问各树：对 WSS_n 而言，当前值为 i，则访问 Tree_{n-i}。

(3) 从 WSS_n 所有值为 i 的序列来看，对应到 Tree_{n-i}，又按照 RRR 的方式访问叶节点；该访问次序在 G3 中有一个序列 TSS_{n-i}。那么可以为每棵树设置一个指针，指示 TSS_{n-i} 序列。

G3 比之 SmRR，访问序列更为平滑，对同一流的两次调度间，相隔比较均匀。G3 比之 RRR，不仅减小了树的规模，将其分割为若干小树，而且用 TSS_{n-i} 的序列结构改进了访问方式，因而是 $O(1)$ 的算法。

4. 公平调度算法类 GPS

1）基本概念

GPS 思想映射到实际算法中时，有一些专业术语，罗列如下。

（1）虚拟时间。在 GPS 情况下，每个队列中的网包到达后，按分配给它的带宽份额进行传输，在此带宽份额下，网包有一个传输开始时间 t_s，一个传输完成时间 t_f；然而由于实际的传输系统中，网包是一个一个传输的，在每个网包传输时，将占据整个管道的流量，无法用连续的理想流体模型来建模，为了逼近 GPS 的情况，希望它尽可能在$[t_s, t_f]$内完成传输，因此提出了一个逼近模型——PFQ。在理想流体模型下，为了方便地评估队列的服务份额，以服务量代替开始时间和结束时间；这就是虚拟时间。

（2）Head packet of a flow。队列头的网包。

（3）Backlogged flow。如果流中有还未接受服务，但已经有从虚拟时间上来说可以接受服务的包（或称已经启动的包），那么该流被称为是积压的。

（4）Service guarantees。实际的调度系统和理想流体系统在性能方面的差值。一般评价指标有以带宽为指标的 B-WFI（Bit Worst-case Fair Index）和以时延为指标的 T-WFI（Time Worst-case Fair Index）。此外评估公平性还有一个指标 SFI（Service Fairness Index），它用来比较在同一时段内，任意两个流所接受的服务量的差值。

2）PFQ 模型

理想流体模型（Generalized Processor Sharing，GPS）是这样的模型：每个输入队列有一个权值，表示在传输链路中所占的带宽份额。数据在传输的过程中，在任何时刻 t 看去，传输链路中按每个需要传输数据的队列所需份额的比例传输数据：

$$r_i = \frac{\Phi_i}{\sum\limits_{\text{active}j} \Phi_j}$$

GPS 模型是一个理想化的流模型，PFQ（Packet Fair Queuing）算法模型则是把该模型应用于网络系统的一类近似算法，也就是当业务流不能无限分割，而只能以网包为最小单位的情况下：在网包进入各自队列时，为其计算在理想流情况下系统在传输前的服务量和传输后的服务量，作为流的标记，这里称为虚拟时间戳——虚拟开始时间和虚拟完成时间。

当上一个网包传输完毕后，下一个网包的选择策略有 3 种：最小虚拟完成时间优先（Smallest virtual Finished time First，SFF）、最小虚拟开始时间优先（Smallest virtual Started time First，SSF）和最小合法虚拟完成时间优先（Smallest Eligible virtual Finished time First，SEFF）。合法是指网包的虚拟开始时间不大于当前的系统虚拟时间。前两种策略对网包的选择只使用一个时间标签，因而可能会产生与 GPS 模型较大的偏差，影响系统的公平性。

3）WFQ ✍WF2Q ✍WF2Q+ ✍SCFQ ✍SFQ

不同的 PFQ 类算法对虚拟时间的计算方式不同，或者对下一个需要调度的网包的选择方式不同，如表 2.1 所示。在具体算法的选择策略中，WFQ（Weighted Fair Queueing）、SCFQ（Self-Clocked Fair Queueing）使用 SFF，STFQ（Start-Time Fair

Queueing)使用 SSF,WF2Q(Worst-Case Fair Weighted Fair Queueing),WF2Q+ 和 QFQ 使用 SEFF。而虚拟时间的计算方式上,对 WFQ 和 WF2Q 而言,流 i 的第 k 个网包的虚拟开始时间 S_i^k 和虚拟结束时间 F_i^k 计算如下(其中 L_i^k 是该网包的包长,ϕ_i 是该流的权值,也可以理解为传输速率,$V(a_i^k)$表示网包 a_i^k 到达时的虚拟时间):

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}$$

表 2.1　算法比较

	WFQ	WF2Q	WF2Q+	SCFQ	SFQ
虚拟时间计算方式	每次有包进出时,复杂	每次有包进出时,复杂	每次有包进入时,简化计算	当前正接受服务的网包的虚拟结束时间	当前正接受服务的网包的虚拟结束时间
选择网包策略	SFF	SEFF	SEFF		

那么虚拟时间的计算如下:

$$V(0) = 0$$

$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{i\in B_j} \phi_i}, \quad t \leqslant t_j - t_{j-1}$$

事实上,虚拟时间是一个离散的过程,每当有网包到达或离开时,会记为事件 j,并计算该时刻的虚拟时间 $V(t_j)$。

WF2Q+则计算如下:

$$V(t+\tau) = \max(V(t) + \tau, \min_{i\in B(t)}(S_i(t)))$$

SCFQ、*SFQ* 则以当前正接受服务的网包的虚拟结束时间作为当前的虚拟时间。

也就是说,在 t 时刻,在所有队列都没有积压网包的时候,把将来最早启动的某一个虚拟时间作为该时刻的虚拟时间;而该时刻如果有积压网包,那么在下一更新虚拟时间的时刻,得到的虚拟时间为 V(t)+τ。

WFQ:给每个包一个 GPS 的结束时间,每次选择等待队列中最小的一个进行调度。可能导致极度不公平。实现复杂。

WF2Q:为了解决 WFQ 的不公平问题,在 GPS 时间下,选择系统中等待的、已经启动的网包中虚拟完成时间最早的。WFQ 中,则是选择在系统中等待的网包中完成时间最早的。

WF2Q+:是 WF2Q 的一个简单的实现,有比较简单的虚拟时间的计算方式。

WFQ 和 WF2Q 都只能在理论上实现,WF2Q+ 难以在高速网络中实现。虚拟时间

的计算和每发送完一个包后需要对下一次调度的网包重排序，所以调度算法的时间复杂度较大。

这是公平调度算法的一种。

4）QFQ（Quick Fair Queueing）

之前提到的公平调度算法都是理论算法，或者无法在高速网络中实现，QFQ 算法则易于实现。它不仅有 $O(1)$的时间复杂度，而且用简单的指令和数据结构就足以实现，适合于硬件实现，在时延和公平性方面则秉承了公平调度算法的一贯特征。

该算法设计不算精巧，结构比较庞杂，但作为 WF2Q＋的一种有 $O(1)$时间复杂度的实现，仍有值得借鉴之处，FreeBSD 系统更是将 QFQ 作为备选调度算法之一。

QFQ 算法的作者在深入研究了 WFQ 系列算法之后，对数据流考虑了两个因素：$\frac{L_k}{\phi_k}$和 S_k。将$\frac{L_k}{\phi_k}$相同的数据流分在同一组内，调度时优先组号较小者；这意味着，权重 ϕ_k 越大，被调度的可能性越大。同一组内的网包在更新虚拟时间时，可以同进退。在组内，将业务流按照虚拟开始时间映射到若干桶内，每个桶连接开始时间相同的业务流。

需要调度下一个网包时，从最小组号开始扫描，碰到的第一个有积压网包的组，进入该组，从虚拟开始时间最小的桶扫描起，直到第一个有积压网包的桶，从中取出第一个流的队首网包发送，更新虚拟时间，更新该数据流的虚拟开始时间。继续上一过程。

该算法的缺点也很明显，它将真实情况做了很多模糊化、近似化处理，所以是一种粗粒度的算法，在某些情况下会有突发状况。

5. 公平调度算法类 GPS＋轮询类 RR

1）StRR（Stratified Round-Robin）

该算法的实质是将队列按权值网包轮循：组内采用 DRR 调度方式，组间采用最早完成时间优先（时间间隔小者优先）的方式。

为每个队列 $f_1,f_2,\cdots,f_n$ 设置带宽份额 $r_1,r_2,\cdots,r_n$，R 为总的带宽且 $\sum_{i=1}^{n} r_i \leqslant R$。设权值 $w_i=\frac{r_i}{R}$。

将队列按 w_i 的值分类，队列 i 属于第 k 类当且仅当$\frac{1}{2^k}\leqslant w_i<\frac{1}{2^{k-1}}$，记该类的所有队列的集合为 F_k，满足 $F_k=\left\{i:\frac{1}{2^k}\leqslant w_i<\frac{1}{2^{k-1}}\right\}$。

先进行组间调度：将带宽分时间片，在实际调度的过程中，在有网包等待传输的情况下，属于第 k 类的队列每 2^k 个时间片调度一次；当出现时间片冲突时（即多个队列同时需要下一时间片），优先选择组号较小、调度间隔较小的队列进行服务。这也就是最早完成

时间优先的意思。

再进行组内调度：类似于 DRR，以字节为单位为每个队列分配一个带宽份额 $c_i=2^k w_i L_M$（这里 L_M 指最大包长），同时为每个队列维护一个计数器，初始化为 0。每次轮循至该队列时，先将计数器的值加上 c_i 的值，如果待发送包的包长小于计数器的值，则发送该包，同时将计数器的值减去包长值；如果该队列还有网包要传输，那将继续传输直到待发送包的包长大于计数器的值，或者直到没有网包要传输。如果待发送包的包长大于计数器的值，则从组间调度处重新开始。

StRR 与 DRR 不同的是，StRR 比 DRR 更好地保证了调度的公平性。而且与 SmRR 相比，它将最坏情况延时降低到一个很小的常量值。在实际应用中，可以使用 bitmap 提高该算法实际应用中的速度。

2）GRR（Group Round-Robin）

顾名思义，这是一种网包轮循：组间可采用各种 PFQ 算法或类 PFQ 算法，组内采用按比例轮循。这是对 StRR 的推广。具体来说，该算法步骤如下。

（1）先为每个队列赋一个权值（非负整数），按权值将队列分组：权值为 φ 的队列分在第 σ 组，$\sigma=\log_2\lfloor\varphi\rfloor$；换言之，第 σ 组包括权值为 $2^\sigma, 2^\sigma+1, \cdots, 2^{\sigma-1}-1$ 的队列。

（2）然后计算每个组内的所有队列权重之和，记为 Φ_σ，$\forall\sigma$。

（3）在实际调度的过程中，先根据 Φ_σ 进行组间时间片分配，再根据 φ 进行组内队列的调度。

① 执行组间调度的时间片分配时，采用 WFQ、WF2Q、HS、SCFQ、SFQ、SRR 等算法，以 Φ_σ 为参数进行时间片分配。

② 执行组内调度的时间片分配时，算法考虑了一个衡量指标$\frac{\varphi_i}{2^\sigma}$，易得 $1\leqslant\frac{\varphi_i}{2^\sigma}<2$，这里 i 是所有队列的队列编号，取非负整数 $1,2,\cdots,n$。不妨单独考查分配给该组的时间片序列。这些时间片按队列的权值等比例地分配给每个队列。直观地看，对于队列 i，设定一个份额值 D_i，初始值为 0。每轮询一周，份额值增加$\frac{\varphi_i}{2^\sigma}$；队列 i 每传输一个包，将份额值 D_i 减 1。由于$\frac{\varphi_i}{2^\sigma}$是不小于 1 且小于 2 的数，因此当 D_i 的小数部分积累到一定量值的时候，必将大于等于 1，此时可以将该队列的网包再传输一次，以达到按权值分配的目的。

在理想情况下，该算法能得到比较好的份额上的公平性和计算复杂度。然而，该算法基于静态信息分配时间片，一种比较恶劣的情况是，某个组根本没有网包可传输，却仍然为其分配时间片，这将造成带宽上的浪费。

3）BSFQ（Bin Sort Fair Queuing）

该算法也是一种网包算法，它为每个待传输的网包赋予一个结束时间，将结束时间相

近的分在一个组(bin)中,每次调度时,在所有 bin 之间选择结束时间最早的,bin 内采用 RR 算法。

6. 基于服务曲线的算法

服务曲线是以时间为参数的服务描述函数曲线。服务描述函数能把对服务质量的要求通过简单的描述函数表示出来。这样对每一种业务类赋予一条服务曲线,指定它每时刻应该收到的最小服务量。

基于服务曲线的算法是一种很好的想法,对不同的业务流设定不同的服务曲线,实际服务量以服务曲线为最低保证。不同的服务描述函数对 QoS 的侧重点不同,而不同的服务描述函数亦能够生成不同的算法。

服务曲线类的 SCED(Service Curve-based Earliest Deadline)算法将到达曲线平移最大延时得到服务曲线,因此该服务曲线可以保证业务的延时特性。具体做法是在网包到达时,计算最晚被发送时间,按照该时间的先后顺序依次发送各个网包。该算法能保证所有服务曲线的前提是所有服务曲线之和不大于系统所能承受的总的服务曲线。而且该算法不能保证业务间的公平性。

而 HFSC(Hierarchical Fair Service Curve)算法则给不同时延要求的业务类分配不同的服务曲线:时延要求严格的业务分配凸的服务曲线,时延要求宽松的业务分配凹的服务曲线,这样可以优先小时延业务接受服务或高速服务。

事实上,GPS 模型的算法也是基于服务曲线的算法原理。它的服务曲线是一条过原点的线性服务曲线,保证了业务速率的要求,缺点则是将带宽和时延的限制互相耦合在一起。

一种比较全面的做法是设计非线性服务曲线,实现时延和带宽的解耦,实现灵活的资源管理和较高的资源利用率。

7. 整体网络流量调度

以上是单个节点调度算法,下面讲网络调度的策略。

1) 核心无状态算法

网络中有一些节点的流量较大,甚至引起突发,这些节点在该类算法中被称为核心节点。与之相对应的是边缘节点。这种区分类比于中国铁路网,徐州、郑州等主干交汇点即可视为核心节点;各铁路沿线的小站则可视为边缘节点。

调度的处理需要进行流分类,为每个流维护一些状态信息,为网包打上调度策略所需的信息标签等操作,其中某些操作只需处理一次,如果这些操作在核心节点处完成,势必造成瓶颈,影响网络的整体性能。那么在该算法中,刻意将这些操作放在边缘节点处完成,生成信息标签,其中一部分信息在传输过程中保持不变,另一部分则在核心节点处网包转发之前进行修改。

该算法的优点显而易见，亦即算法初衷，降低了算法复杂度；缺点是算法的有效运行依赖于边缘节点和核心节点的配合，需要修改网包头（附加信息标签），可能会影响某些协议。

常见算法有 CSFQ（Core Stateless Fair Queuing）、CJVC（Core stateless Jitter Virtual Clock）。

事实上，网包调度算法的目标既要考虑算法的计算复杂度和空间存储复杂度，又要尽可能满足不同队列的要求。那么在不同的系统下，不同的算法可能有不同的效果。

2）结合缓冲管理的算法

根据机理的不同，队列管理是流量控制的关键，而队列管理细分为缓冲管理和网包调度，如图 2.15 所示。如前所述，网包调度能够影响对业务流进行服务的各种 QoS 指标，缓冲管理亦如此。综合考虑两者的影响，得到了目前已经提出的 C-DBP-Delay-Loss 和 JoBS 等算法。事实上，由于网包调度算法主要考虑的 QoS 属性包括带宽利用率、时延衡量指标，一旦综合考虑缓冲管理，还需要加上丢失率等因素。

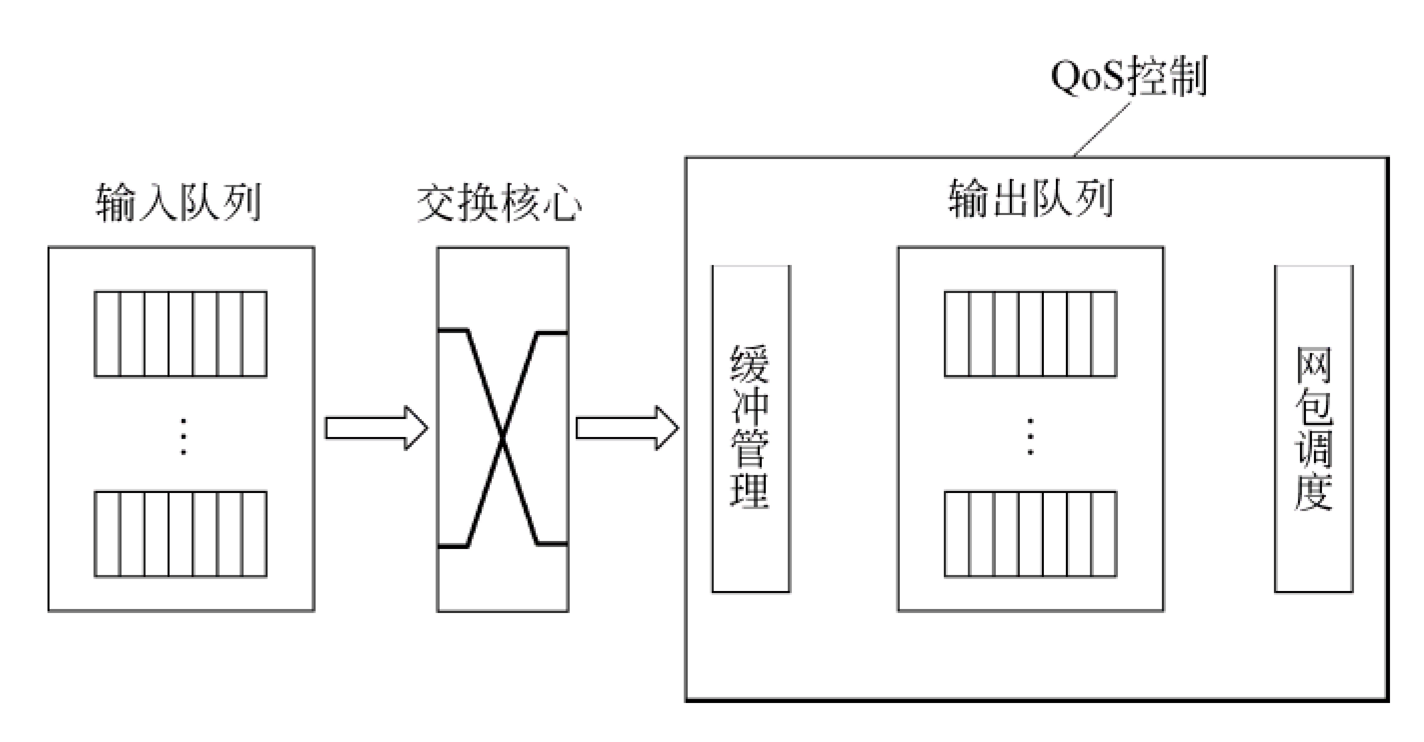


图 2.15　输入输出排队路由器中的网包处理流程

C-DBP-Delay-Loss 算法希望通过综合考虑时延要求和丢失率要求，区分各个业务流，给不同 QoS 要求的业务流提供不同级别的服务。在时延控制方面，采用(m,k)模型，即在给定流的每 k 个连续网包中，至少有 m 个网包能满足其端到端的时延限制。在丢失率方面，采用了 DBP(Distance Based Priority)算法的思想，同样在给定流的每 k 个连续网包中，统计到达某失败状态（也就是当发送网包数小于丢弃网包数）所需发送的网包数目，该数目越大，优先级越低；系统的调度策略即基于优先级的网包调度。

JoBS 算法主要是为各个业务流分配服务速率，以保证时延和丢失率。具体流程如下：当有网包到达时，为所有队列计算时延估计，然后调整速率分配，以保证所有 QoS 约束。但如果无法满足所有要求，速率分配将被归结为多目标优化问题：约束是关于时延、丢失率的 QoS 约束，链路和缓冲容量的系统约束；目标函数既需要保证丢失率最小，又需要尽可能保持当前速率分配不变，后者是为了避免服务速率抖动。这一算法的缺点也是

基于速率分配算法的共同缺点——速率和时延的约束是耦合在一起的。

3）分层链路共享算法

很多调度算法都是在流之间进行的，各个流之间是共同竞争的关系，而该类算法对链路的带宽分配方式分层，为不同层的各个元素分配不同的带宽，常见的算法有 CBQ（Class-Based Queueing）、H-PFQ（Hierarchical Packet Fair Queueing）和 HFSC（Hierarchical Fair Service Curve）。

提出分层的共享算法是因为链路的共享存在多种情况，比如一条链路可以被多个公司、组织共享，也可以被多种协议或者多个应用共享。随着网络规模的发展，单一层次的链路共享方式已经不能满足复杂的系统需求。因此出现了分层的共享算法。这类算法把不同类型的网流划分成不同级别的类，并同时考虑流对实时性和链路共享的需求，按照不同的层次进行网包调度，从而提供了在分层共享数内进行不同等级的共享链路带宽资源的方法，如图 2.16 所示。

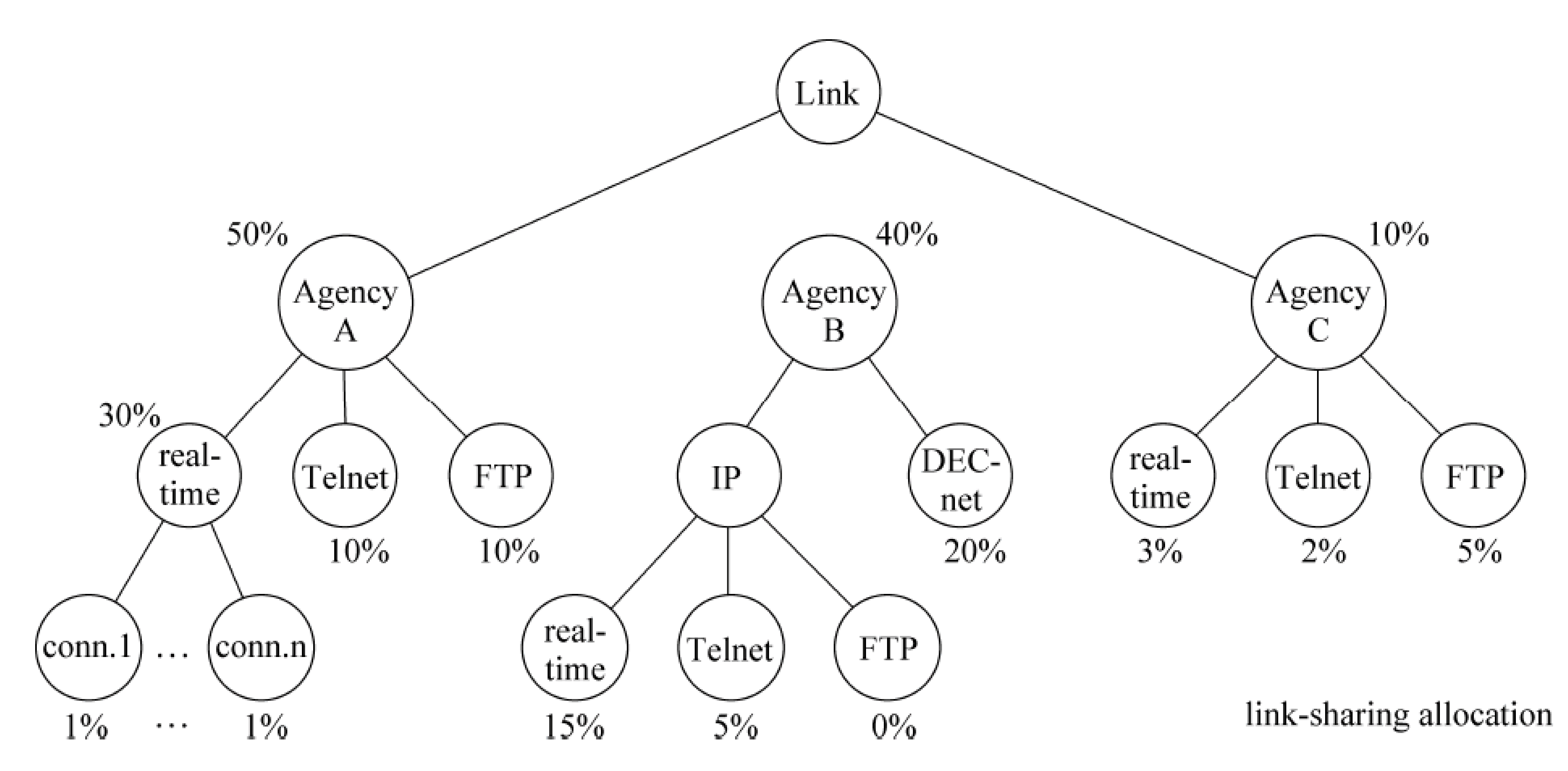


图 2.16 分层共享算法

分层共享算法的目标是，如果某个业务有数据到达，系统应该能够为其提供预先分配的保证带宽；如果某个类的未用完带宽可以根据一定的规则被同级别的其他“兄弟”共享。为此系统一般设计两级调度器：常规调度器和链路共享调度器。当系统没有出现拥塞时，使用常规调度器，使各个业务能够获得所需的服务。当系统出现拥塞时，启动链路共享调度器，对某些超出额定带宽的业务进行限制。这类算法的优点是在保持高链路带宽的情况下提供很好的带宽、时延和公平性保证。

4）比例区分算法类

该算法类基于 DiffServ 架构，借鉴了 GPS 的思想，借鉴了 PQ 的思想，其中部分算法借鉴了基于时延调度的思想。先介绍 3 个概念：相对区分服务、比例区分服务和比例时

延区分服务，它们之间的关系如图 2.17 所示。

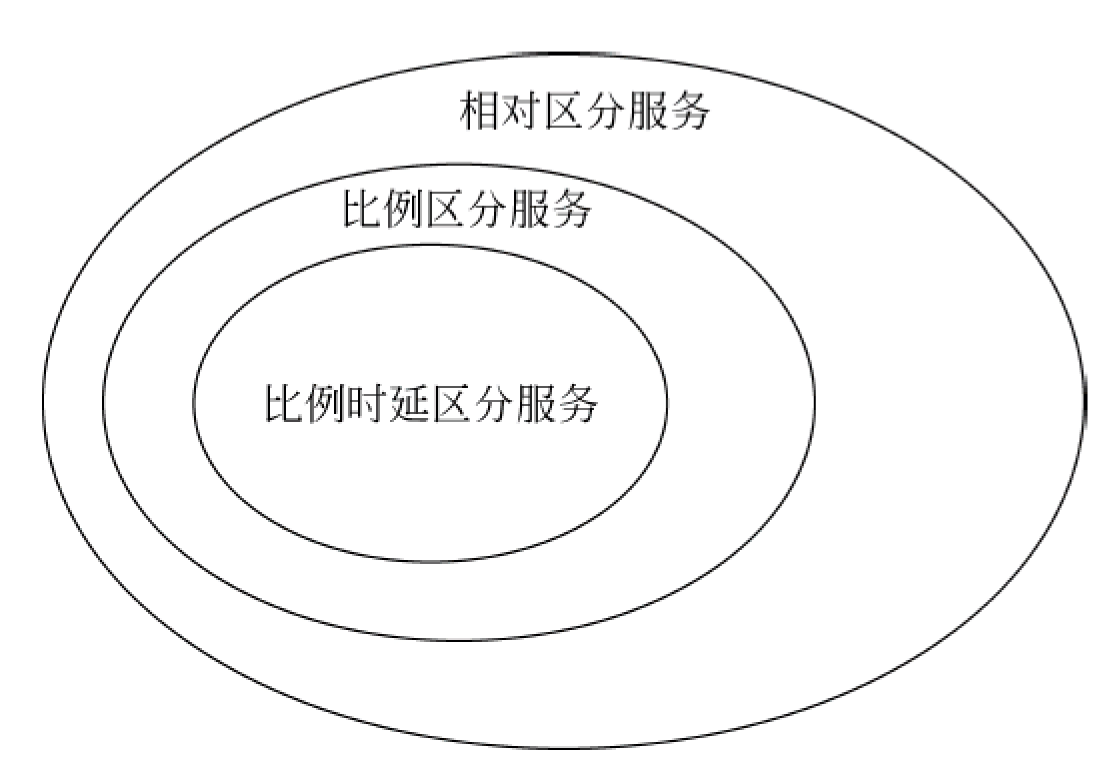


图 2.17　区分服务的概念关系

首先是相对区分服务的设计。将不同的数据流组合成服务类，服务类按照服务质量进行排序，优先服务高级别的类。用户可以根据质量要求和价格选择服务类。这实际上是 DiffServ 的设计思想。

比例区分服务模型则使得任意两个数据类在每一跳获得的服务满足由网络管理者设定的比例。它也是一种相对区分服务。

比例时延区分(Proportional Delay Differentiation，PDD)模型则给出任意两个数据类的时延比例，以保证时延要求小的数据流优先获得服务。WTP(Waiting Time Priority)、PAD(Proportional Average Delay)、HPD(Hybrid Proportional Delay)等算法皆如此原理。不同的是，WTP 算法考虑的时延是队列队头网包的等待时间，PAD 算法考虑的时延是已发送网包的平均等待时间，HPD 算法是 WTP 算法和 PAD 算法的结合，也就是对队列队头网包的等待时间和以发送网包的平均等待时间进行加权和，优先考虑该和按比例较大的队列。

以 WTP 算法为例，设 $w_i(t)$是 t 时刻，队列 i 队头网包的等待时间，该网包所属数据类的比例系数是 c_i，计算规格化等待时间 $\tilde{w}_i(t)=w_i(t)/c_i$，每次调度时，选取非空队列中 $\tilde{w}_i(t)$最大的队列进行调度。

显然，WTP 算法在负载较轻时，容易偏离理想 PDD 模型。PAD 算法在数据流突发时，容易偏离。两者的结合在性能上会好很多。

8. 结论及展望

本节对已有的网包调度算法做了调研，在深入分析其原理和动机之后，将其分为轮询类、公平调度类、两者结合类、服务曲线类、考虑整体网络类等大类。但由于互相之间互为借鉴，如部分公平调度算法可以用服务曲线来描述，分层链路共享算法可以在各层间借鉴 GPS 的思想，比例区分算法类又可以应用于缓冲管理、分层链路共享等算法中。又比如

很多算法类综述中都会提及基于时延的算法类，那么这里将部分基于时延的算法按照算法本质归于其中的某些类。由于不同算法的侧重点不同，所保证的性能不同，在具体选择时无法一概而论，需要结合具体需求。

在这里提及的这些大类的方向上，可以做一些细微的工作，比如为单节点处的GPS类调度算法建立不同的服务曲线。此外，对于各种调度算法，另一个可做的工作是对它们的实现和基于实验的性能比较；以及进一步地建立服务曲线以比较之。那么由于调度算法的工作在现有的机制方向上，已无大的突破，也许需要考虑诸如能耗低碳、绿色计算等新的评价指标以提出新的思路，以及需要新的思想、新的设施的提出。这些工作亟待以后的研究。

2.3.4 现有流量管理系统

基于目前网络环境的需要，业界也有诸多公司和组织在进行着这方面的研究，他们的产品既有开源的流量管理系统，也有商业化的系统。这些系统各有优势，也在一些方面存在着不足，下面就对当前比较流行的几种网络流量管理系统进行简单介绍。

1. 应用协议识别

基于深度检测的协议识别不仅检测网包头部信息，而且扫描网包载荷(Payload)以能够识别隐藏的协议特征，从而准确识别出流量的协议。由于正则表达式具有更强的表达能力和灵活性，目前相关的研究比较倾向于使用正则表达式。较为流行的几种工具有IPP2P、OpenDPI和L7-filter等，而其中L7-filter的应用最为广泛。

L7-filter(Layer 7 Packet Classifier)是基于Linux的Netfilter系统上实现的基于网包载荷的协议识别工具，其处理流程如图2.18所示。

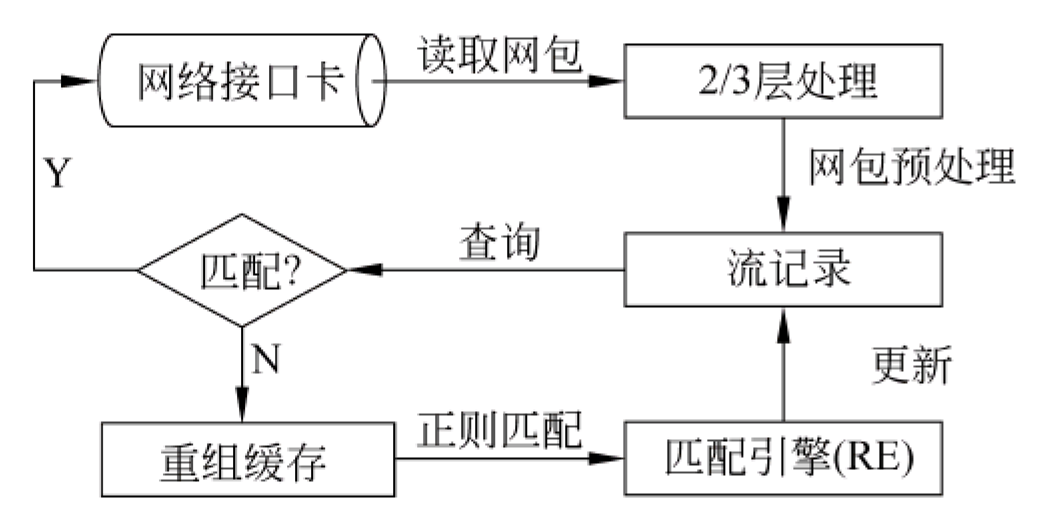


图2.18 L7-filter处理流程

L7-filter根据各种协议RFC标准，并通过分析实际流量，提取协议特征，写成正则表达式。如QQ协议的正则表达式为“^.?.?\x02.+\x03$”。网包中第一个、第二个或者第三个字符为0x02，最后一个字符为0x03匹配这条表达式。L7-filter中每个协议的特征正则表以.pat文件的格式存放，至2009年5月28日的版本已经支持113个协议。随

着一些软件的更新，L7-filter 的特征表达式也要不断地更新，用户也可以根据自己的实际环境，添加、删除或者更新的正则表达式以满足自己的需求。在匹配过程中，所有要查询的协议特征组成一个单链表，到达的网包从链表头开始遍历，直到查询到最终结果或者查询到队列尾部。

为提高 L7-filter 的准确率，L7-filter 以“连接”为单位进行匹配。每个连接在系统中有一个对应的重组缓存(Reassembling Buffer)，到达的网包经过预处理进入重组缓存，然后缓存触发与特征链表的匹配。匹配引擎返回识别结果后，该连接后续的网包就直接打上了协议标记而不必进行匹配。如果匹配引擎查找失败，则当该连接下一个网包到达后会重新进行匹配，当重组缓存中收到的网包数超过一定阈值(默认为 8，用户可以配置)，该连接被标记为 NO_MATCH，该连接后续的网包也不会触发匹配。其匹配过程如图 2.19 所示。

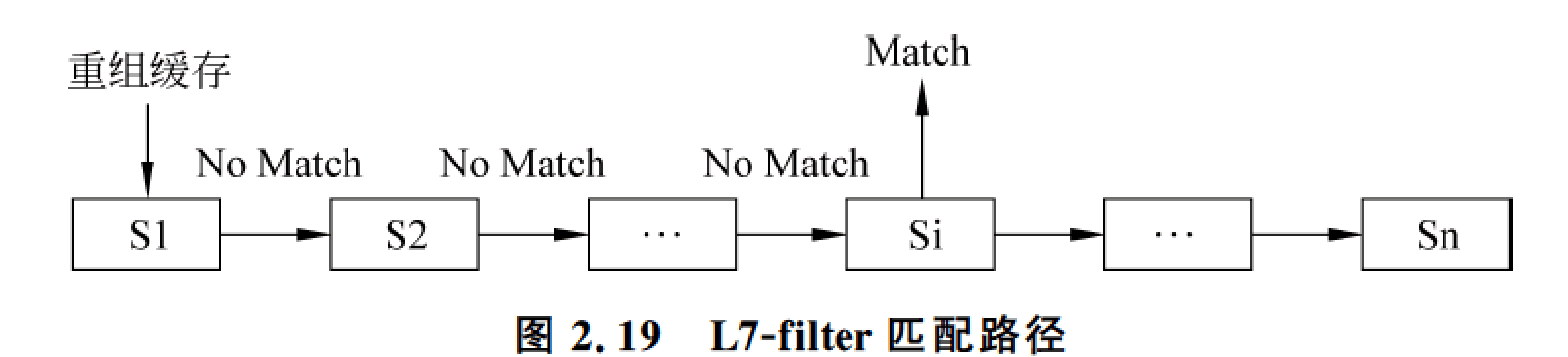


图 2.19　L7-filter 匹配路径

最初 L7-filter 开发者在内核中实现程序，这是因为他们认为正则表达式的匹配在内核中更为高效。而实际上计算密集型的程序并不适合在内核中实现，而且在内核中实现的程序难以并行化。L7-filter 的开发者认识到这点并于 2006 年推出了基于 NFQueue 的用户态的版本。为充分发挥多核平台的优势，可以在用户态程序上进行优化和改进。用户态的 L7-filter 是单进程的，如图 2.20 所示。

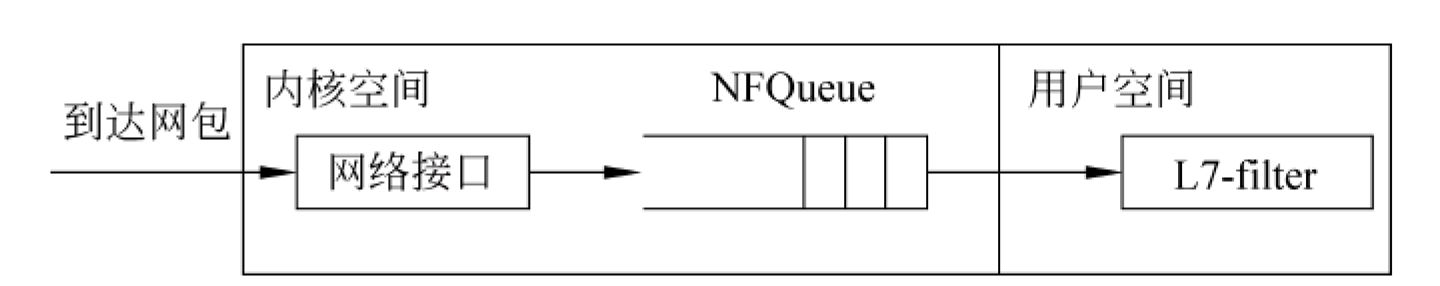


图 2.20　单进程用户态 L7-filter

2. 基于 Linux 系统的流量管理

1）Linux TC

Linux TC 是 Linux 自带的流量控制器，从内核 2.2 开始 Linux 就提供了这项灵活、强大的流量控制功能。其包括运行于内核空间的流量控制系统和运行于用户空间的应用程序。

Linux TC 的流程如图 2.21 所示。整个大方框表示内核空间，主要包括了网包调度机制(Queue Discipline，队列规定)、类(Class)、网包过滤器(Filter)和策略(Policy)4 个部

分。网包从最左面进入设备,进入 Ingress 队列规定,并根据策略放行或者丢弃网包。策略放行且网包是发往本地进程,则进入 IP 协议栈处理并提交给用户进程。如果它需要转发而不是进入本地进程,将发往 Egress。Egress 分类器经过审查,将网包放入若干队列规定中的一个进行排队,即入队。网包进入队列后,经等待内核处理并通过某网卡发送,即出队。

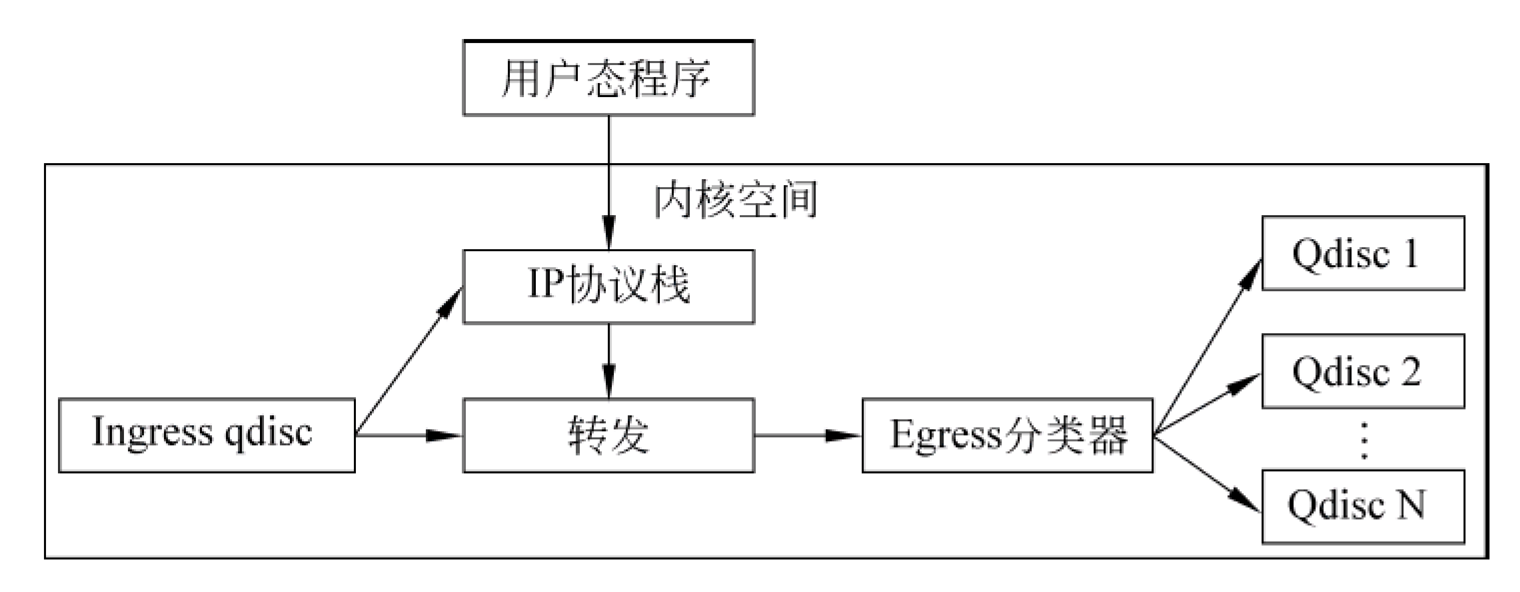


图 2.21 Linux TC 系统架构图

Linux TC 具有以下特点。

(1) 高度的开放性。该系统设计了一个开放的框架标准,根据这个框架标准实现的调度算法可以容易地添加应用到系统中。

(2) 易用性。该系统提供了运行于用户空间的桌面应用程序以对内核空间的流量管理系统进行配置。通过桌面应用程序,用户可以方便灵活地对每个网络应用选择调度算法和配置运行参数。

(3) 成熟度高。该系统中实现了现有的多种分类和调度算法。包括 RED、CBQ、SFQ、TBF 等十几种算法。每种算法都提供了丰富的接口函数,用于对网包进行调度、转发,以及自身的维护和配置。

2) Untangle

Untangle 是一款基于 Debian Linux 的统一安全网关系统,它包含病毒拦截、入侵拦截、防火墙、协议控制等多项功能。如图 2.22 所示,通过启动协议控制的 7 层协议过滤功能,管理人员可以完全掌控 P2P、视频直播、在线游戏等网络滥用行为。通过双向阻止受控软件对受保护网络的访问,封堵 P2P 等开放多个 TCP 端口的应用,以保证带宽;通过阻止防火墙规则无法完全禁止在线视频、在线游戏等应用,来提高工作效率。另外,还可以编写自定义的特征码描述任意协议,协议控制模块通过使用特征码,在所有端口上来界定需要控制的协议。有些协议,如 IM 和 P2P 等,会进行端口跳转,一旦在默认端口被阻断,它们会通过 80 或 25 端口进行连接。而 80 和 25 端口是 Web 和 E-mail 的常用端口,一般不能被禁止。Untangle 的协议控制模块则能够识别这种端口跳转行为,进而阻断并记录网络连接企图。

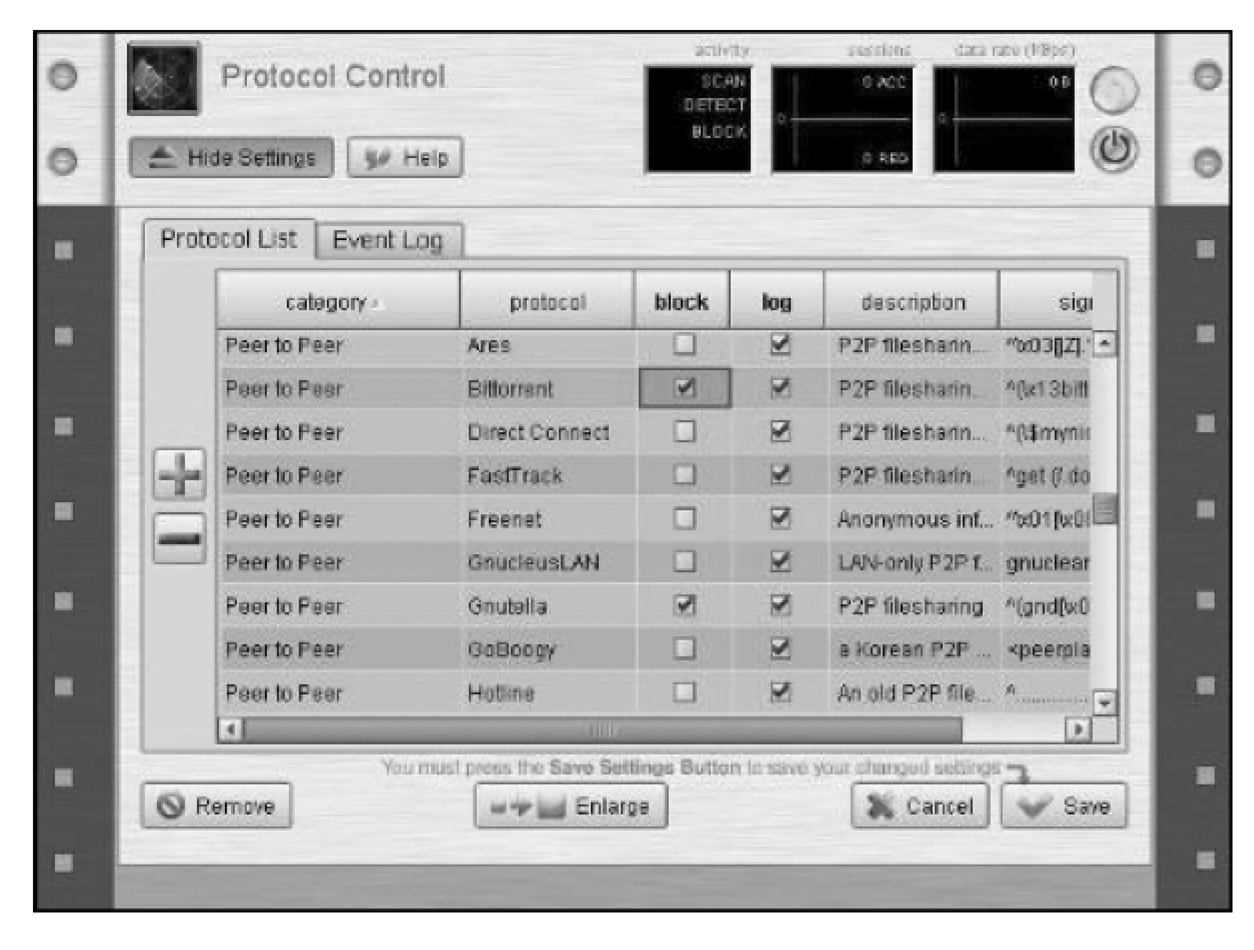


图 2.22　Untangle 协议控制模块

3. 基于 FreeBSD 系统的流量管理

FreeBSD 也推出了类似功能的 ALTQ（Alternate Queuing）。以下介绍基于 FreeBSD 系统的开源流量管理系统 m0n0wall 和 pfSense。

1）m0n0wall

M0n0wall 是基于以性能和稳定性著称的 FreeBSD 内核的嵌入式防火墙系统，它可以提供 VPN、DNS 转发、动态 DNS、流量控制等多项功能，并能专门解决像 ADSL 等非对称链路固有的因上行通道饱和造成 TCP 的 ACK 确认包拥塞所导致的下行带宽利用不足的问题。另外，M0n0wall 的一项非常优秀的功能就是带宽管理 Traffic Shaper 功能，通过策略对带宽进行管理，保障关键服务的带宽，虽然功能设置不是非常丰富，但是对应普通用户来说还是够用的。m0n0wall 项目启动于 2003 年 2 月，于 2004 年 2 月发布版本 1.0。M0n0wall 提供界面相当友好的完全基于 Web 界面(PHP)的配置管理，局域网内任何一台同一网段的计算机通过 M0n0wall 默认 IP 地址登录 Web 界面后均可对系统进行管理。图 2.23 展示了 m0m0wall 管理系统的界面。

2）pfSense

pfSense 项目由 BSD perimeter LLC 公司推出，该项目启动于 2004 年，是 m0n0wall 的克隆项目，它包含所有商业防火墙和 UTM 的功能。pfsense 是基于 FreeBSD 8.1-RELEASE 的软路由服务器平台软件，于 2006 年 2 月发布版本 1.0，目前为版本 2.1。图 2.24 展示了 pfSense 管理系统的界面。

4. 商用流量管理系统

北京派网、宽广电信、畅讯科技等多家公司专门从事流量管理产品的研发，而像 Cisco、



图 2.23　M0n0wall 管理系统的界面

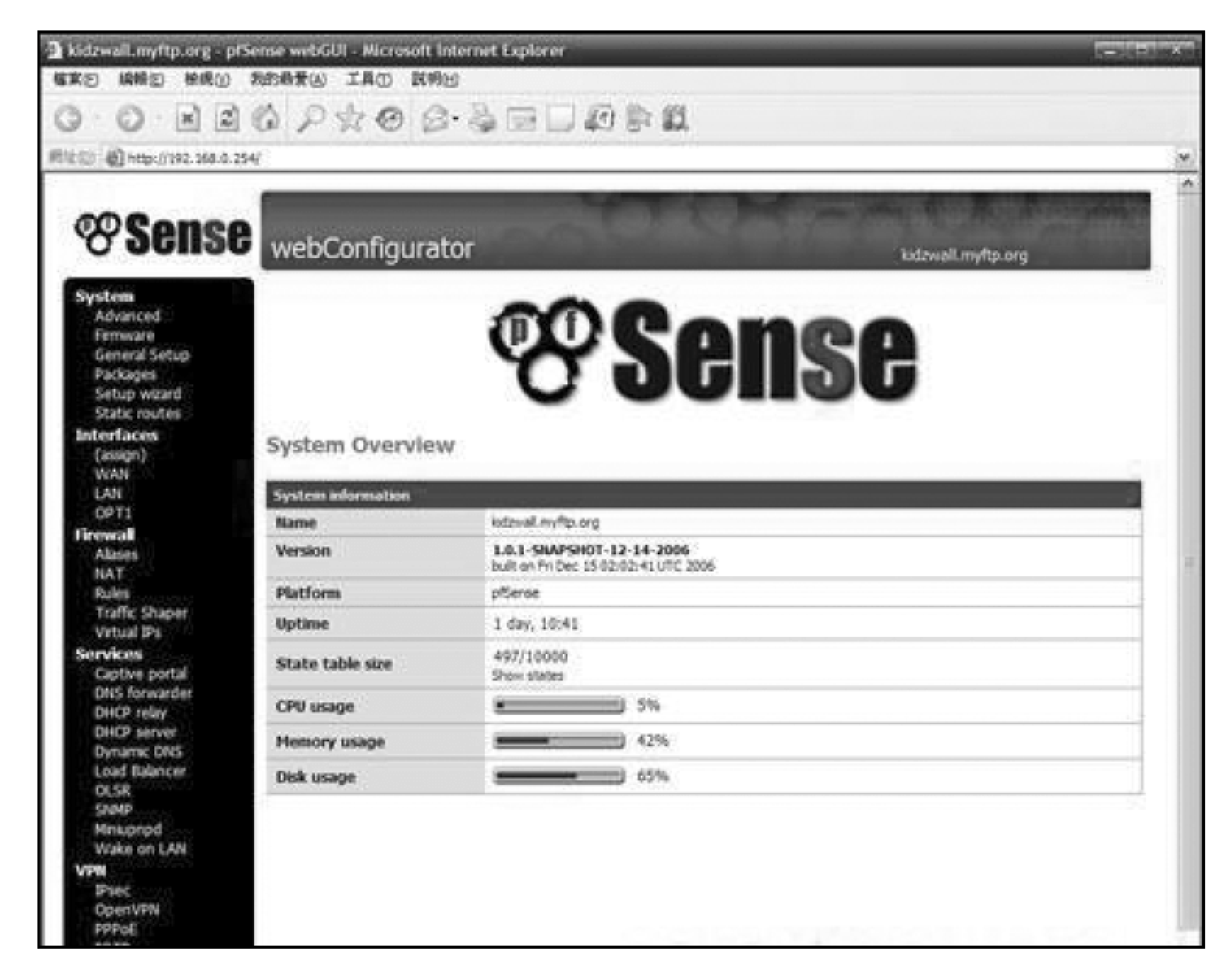


图 2.24　pfSense 管理系统的界面

Juniper、华为等企业也有相关的流量管理设备。其中 Panabit、TMA、QQSG 均是国内比较知名、应用较为广泛的商用系统。

1）Panabit

Panabit 系统是由北京派网软件有限公司出品的网络应用层流量监控管理系统。Panabit 基于 FreeBSD 操作系统开发，特别针对 P2P 应用的识别和控制进行开发与优化。其标准版可以免费使用，但限制并发连接数为 256。其对应用层的协议识别采用深度包检测的方式，对 P2P 协议的识别有较高的识别度和准确度。图 2.25 展示了 Panabit 对整个网络的流量进行监控和分析的情况。

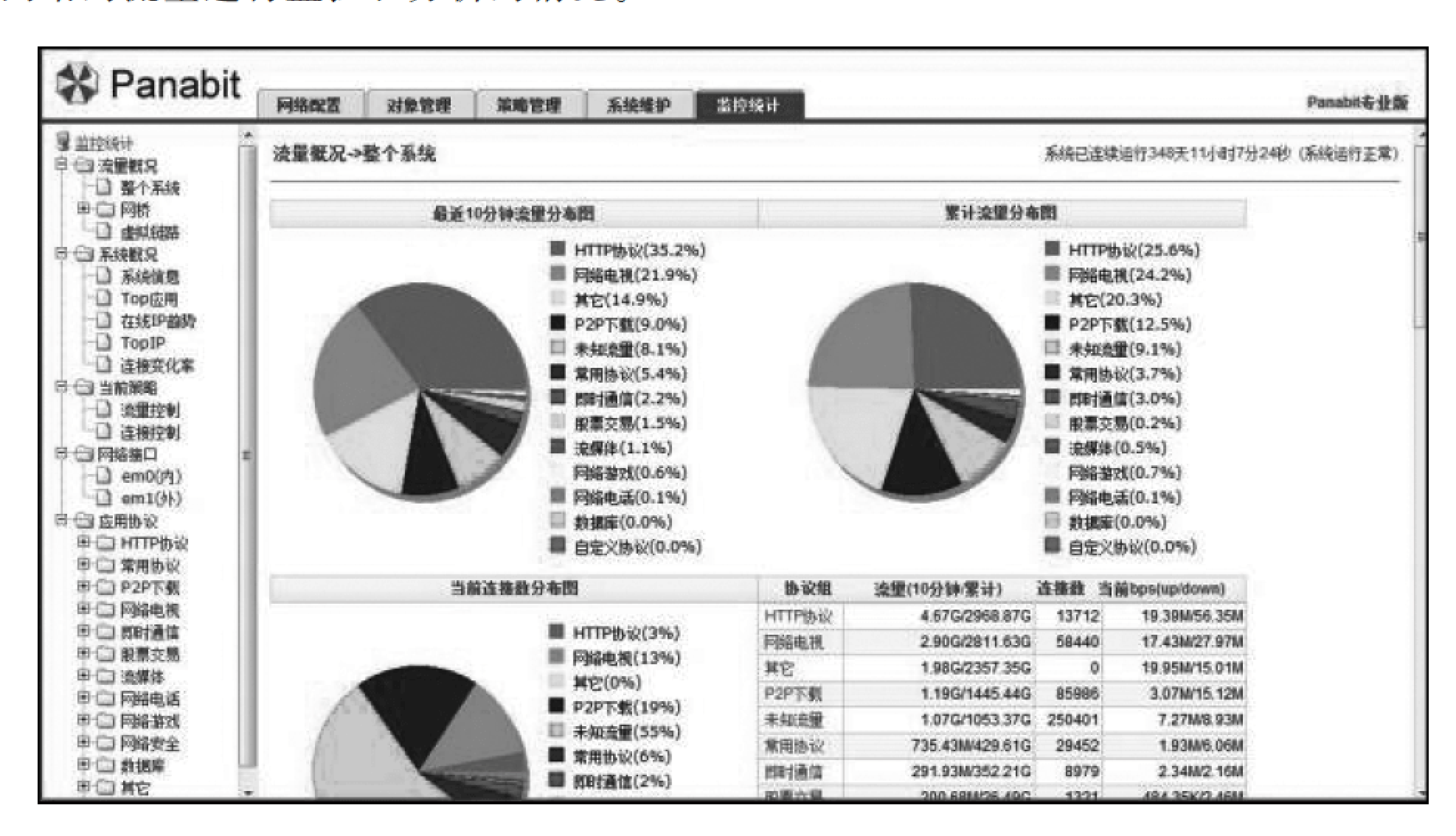


图 2.25　Panabit 整体系统流量展示

2）TMA

TMA(Traffic Monitoring & Administrator)网络流量监控系统是北京宽广电信高技术发展有限公司针对以互联网 IP 技术为核心的各级网络实际运行过程中出现的流量监控和管理问题，开发的具有完全自主知识产权的新一代网络流量监控管理系统，由硬件探针(TMA 1100)、光分路合路器(OptiSwap)和流量监控中心服务器(TMA Server)构成。其特点是综合了 DPI 和 DFI 业务识别算法，能有效识别加密业务，且采用纯硬件架构，提供了很好的性能支撑；支持多维并行流控策略，为用户提供了高精度的灵活流量控制能力。图 2.26 展示了 TMA 监控系统对业务流量进行监控和分析的情况。

3）QQSG

QQSG 网络服务控制管理系统是畅讯科技公司自主研发的一款网络服务管理和保障系统。它采用软件与硬件相结合、高速数据内容检测、数据流状态监测、IP 隧道和微码

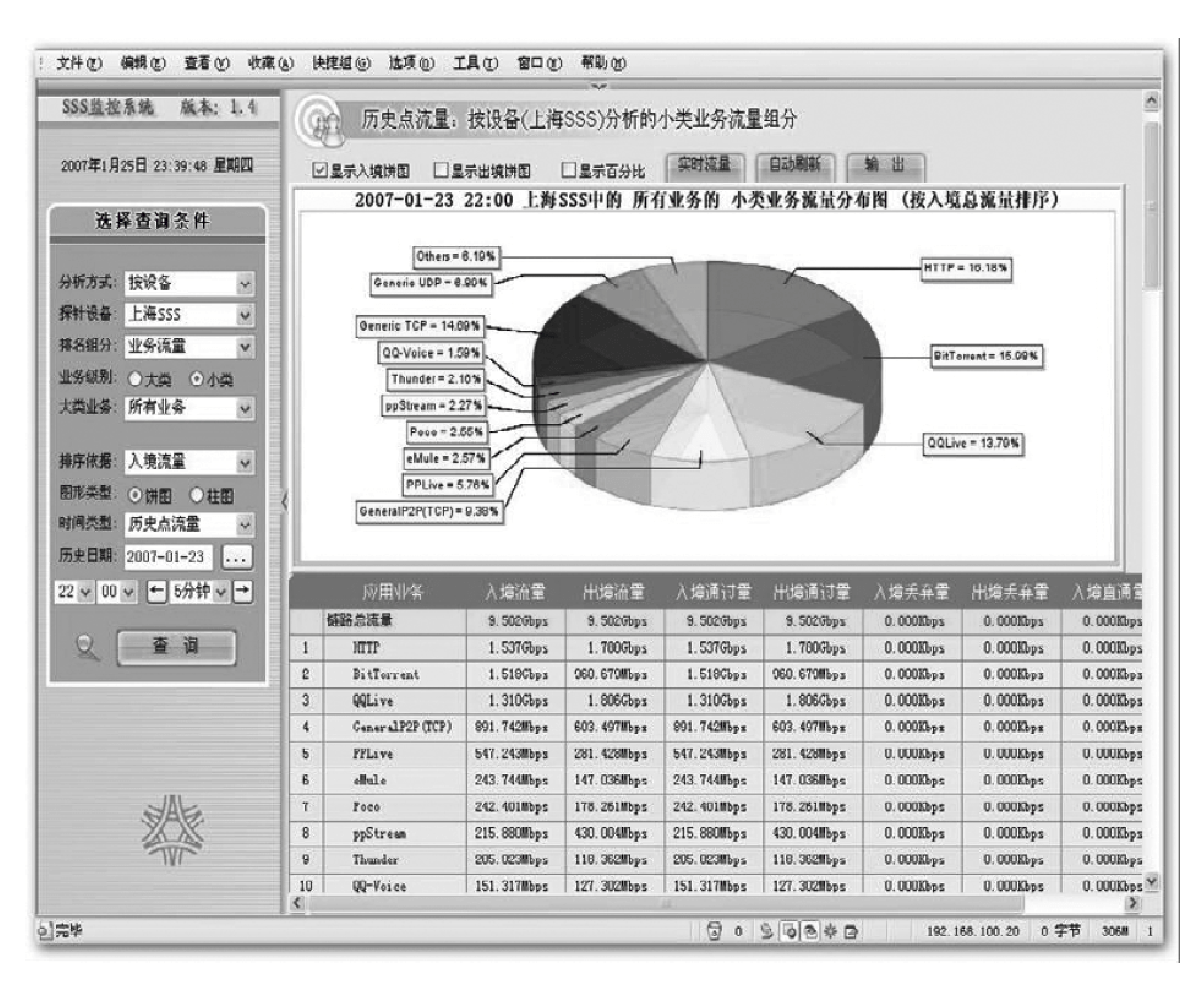


图 2.26 TMA 监控系统

固件的方法,在对网络应用深度解析的基础上,实现了 2~7 层的应用识别分类、流量属性分析、流量策略管理、分类统计报告以及状态预警等多种功能。图 2.27 展示了 QQSG 对全网业务流量进行分析的情况。

5. 小结

虽然当前有多种流量管理系统,然而都尚不完善。开源的流量管理系统虽然有高度的开放性,但是管理操控要求高,协议特征少,限制了其广泛应用。商用的流量管理系统虽然性能有较大提高,管控简单灵活,但价格昂贵,且仍难以满足网络管理系统高性能、高灵活性等的需求。而且,基于端口和行为的两种识别方法准确度较低,不能满足当前精度的需求,而基于 DPI 的识别方法虽然精度较高,但是计算复杂性高,往往需要 TCAM 或搜索处理器等硬件支持。所以,如何充分利用现有的多核平台,设计一种易用、灵活且高性能的软件网络管理系统具有较高的学术和商业价值,也将是高速网络流量管理系统今后发展的一个方向。

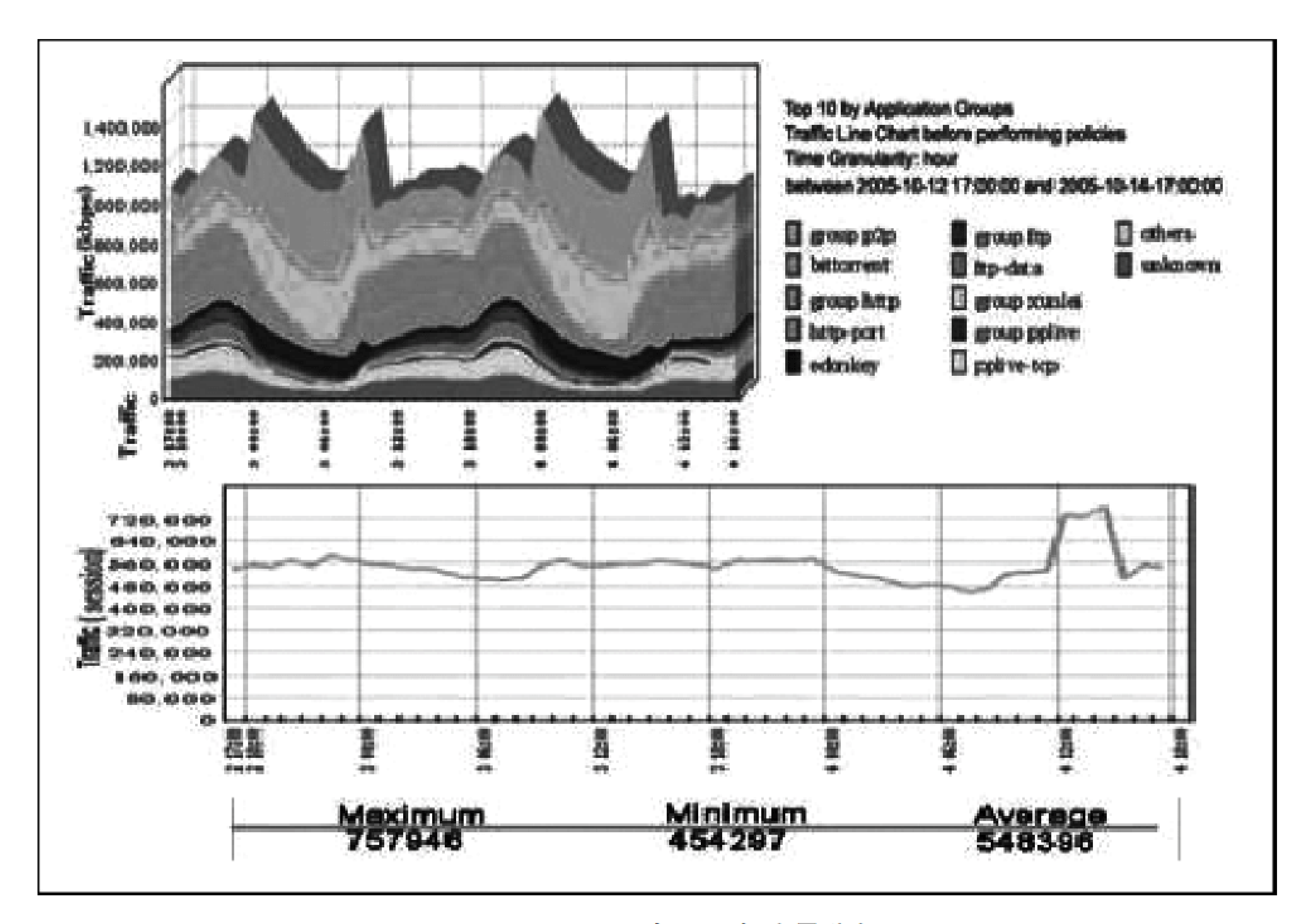


图 2.27　QQSG 全网业务流量分析

2.3.5　协同式流量管理系统

通过分析 DPI 实现协议识别系统，弄清如何在不降低识别准确度的前提下提高系统识别的性能，并结合可信网络连接技术、高速流量记录查询技术和协同交互思想，提出一种协同式的高速网络流量管理系统。协同式流量管理系统采用协同交互思想，将网络内部多个独立的流量管理系统连接起来，对各子网的疑似流量进行统一整合，集中分析，统筹调度。最终将分析结果形成管理策略，按需下发实施。以分布式拒绝服务攻击（DDoS）为例，攻击源通常比较分散，攻击流量的特征不明显，使用单个的流量管理系统，难以对攻击流量进行有效的识别。但是，采用这种协同式的流量管理系统，各个子系统可以将无法识别的疑似流量进行汇总，交由分析控制中心分析其共同特征，有效提高识别效率。

1. 设计思路

Untangle 系统是一款开源的安全网关系统，同时它也有着强大的流量管理功能，其协议控制模块还有着具有高度定制性的独特优势。所以，协同式流量管理系统采用 Untangle 系统作为基本的管理部件，按照协同交互的思想，利用可信网络连接技术，将多个 Untangle 系统连接起来，统一由一台分析控制中心管理。如图 2.28 所示，整个协同系

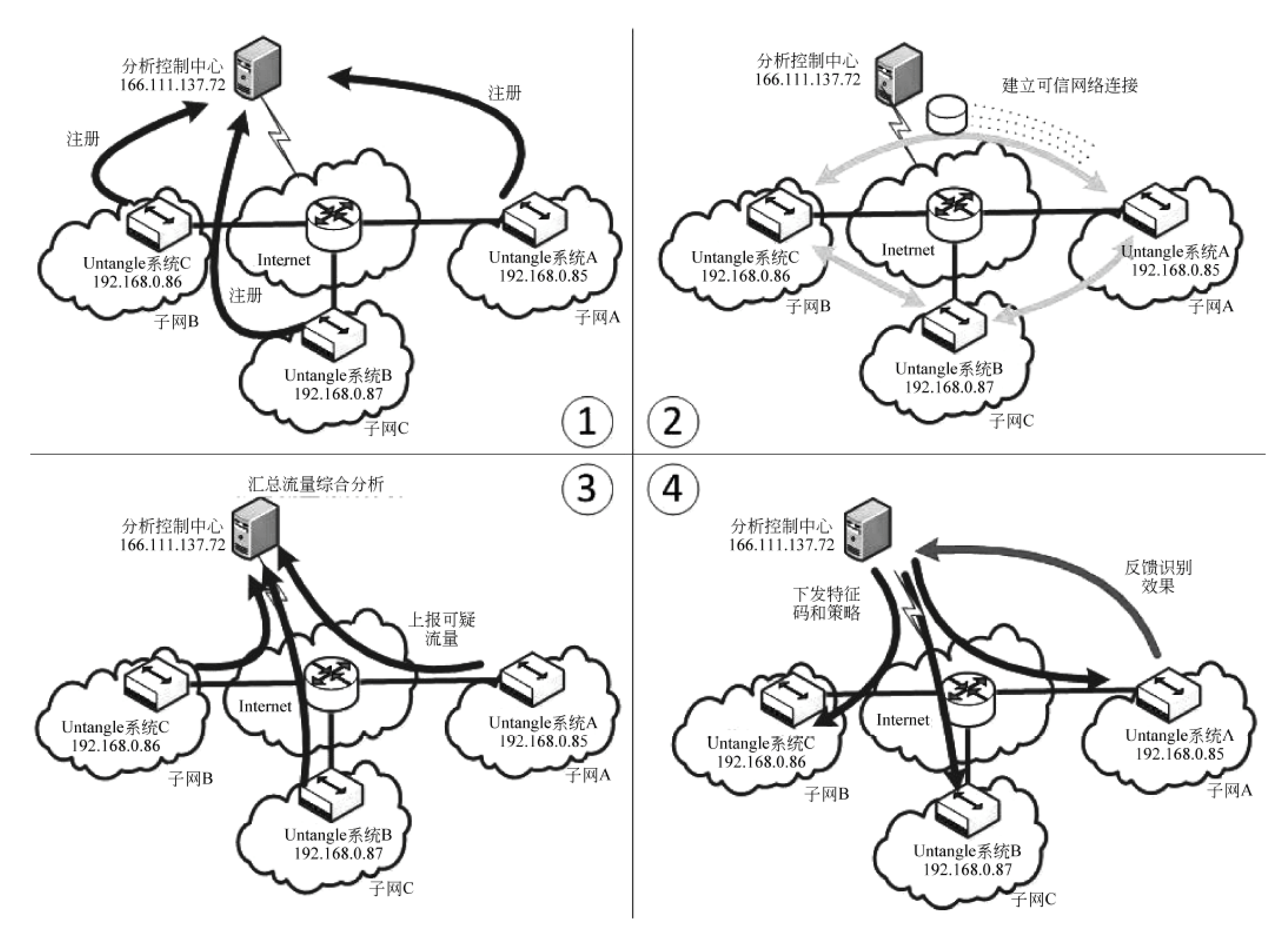


图 2.28 协同系统的 4 种协同行为

统有 4 种协同行为，分别如下。

(1) Untangle 子系统启动后，连接到分析控制中心并进行注册。

(2) Untangle 子系统和分析控制中心之间建立可信网络连接。

(3) Untangle 子系统将未识别的可疑流量上报，分析控制中心汇总进行深度包检测。

(4) 分析控制中心将生成的特征码及策略下发，Untangle 子系统将基于特征码的识别情况向分析控制中心反馈。

2. 系统实现

为实现协同式网络流量管理系统，需要部署分析控制中心，负责对 Untangle 子系统的统一管理。同时，对 Untangle 子系统中的协议识别模块进行改造，使其具备三项功能：一是，将无法识别的疑似流量上报给分析控制中心；二是，将分析控制中心下发的特征码和策略应用到协议识别模块中；三是，将识别结果向分析控制中心反馈。Untangle 子系统上报的疑似流量在分析控制中心进行汇总，通过深度包检测对汇总的疑似流量进行综合分析，生成相应的协议特征库和管理策略，下发到各个 Untangle 子系统的流量管理模

块中。分析控制中心还有另外一项重要的工作就是，将各子系统反馈的特征码和策略的识别效果进行汇总分析，并带入到深度包检测环节，作为辅助参考，加速检测过程和提高检测精度。由图 2.29 可以看到分析控制中心和流量管理子系统相互协调工作的一个基本流程。

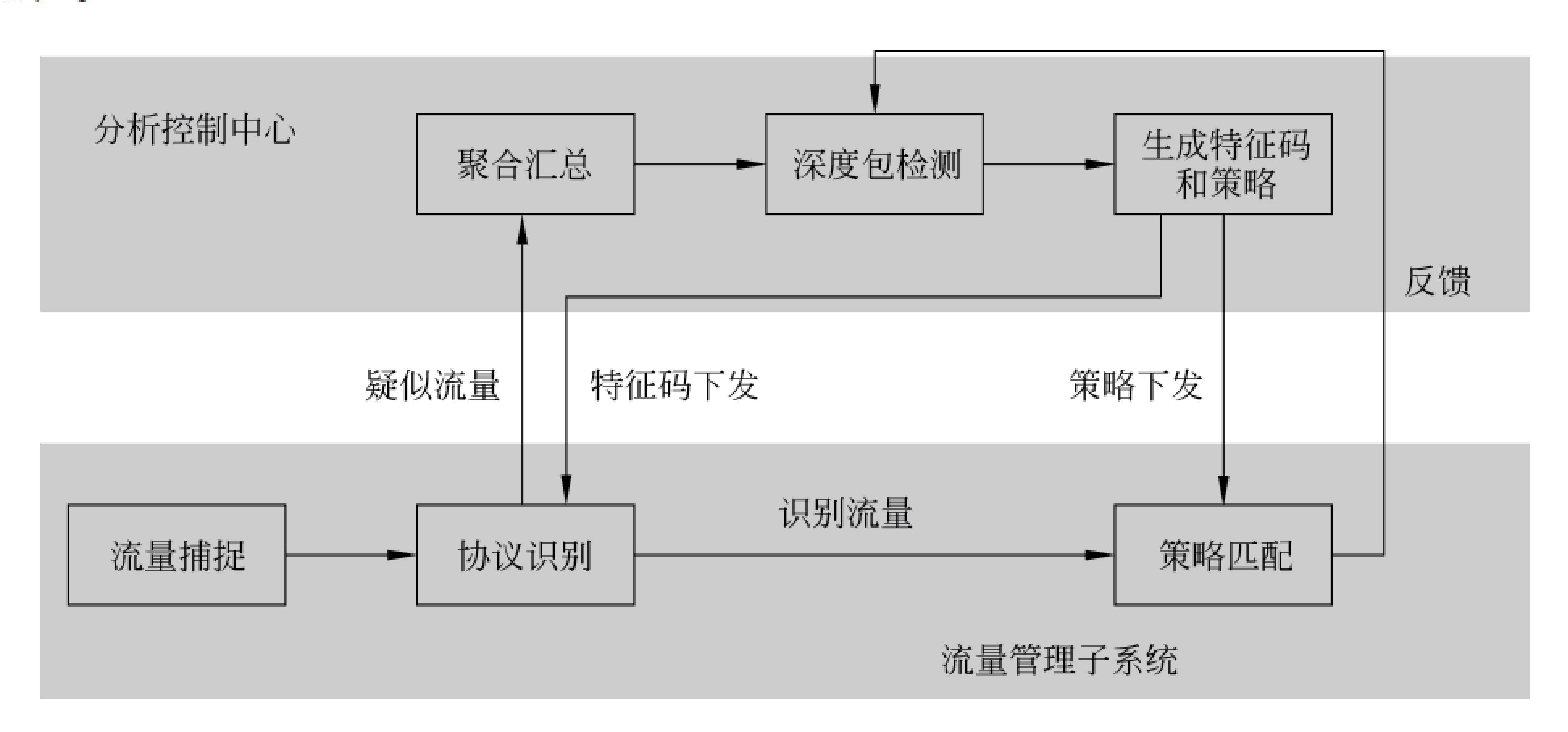


图 2.29　分布式流量管理系统流程

3. 优势评估

将深度包检测从各子系统剥离，各子系统只需根据特征码进行简单的协议识别，降低了各子系统的协议识别压力，提高了子系统的识别性能。单独采用分析控制中心进行专门的深度包检测，将各子系统无法识别的疑似流量进行汇总，并将各子系统反馈的特征码识别情况带入到深度包检测中，形成一个良好的循环机制，以提高深度包检测的准确度。另外，通过分析控制中心对各子系统进行统一管理，使整个系统更灵活、易用，降低了网络管理成本。

2.4　结　束　语

本章首先阐述了网络流量测量和管理的基本概念及其重要性，简单地介绍了现有的网络测量技术、算法和相关研究成果，并介绍了开源流量管理系统和商用流量管理系统的特点及不足。随后，提出了协同式流量管理的概念，介绍了协同式流量管理系统的设计思路和具体实现，并认真评估了协同式流量管理系统的优势。该系统充分利用了可信网络连接技术、深度包检测技术、高速流量记录查询技术等现有技术，提供了一个高效的、实用的、协同式的高速网络流量管理方案。

参 考 文 献

[1] IPFIX. http://www.ietf.org/html.charters/ipfix-charter.html.

[2] PSMP. http://www.ietf.org/html.charters/psamp-charter.html.

[3] Wenjia Fang, and Larry Peterson. Inter-AS traffic patterns and their implications. Global Telecommunications Conference, 1999.

[4] Peter Phaal, Sonia Panchen, and Neil McKee. InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks. RFC 3176, 2001.

[5] Vern Paxson, et al. Framework for IP performance metrics. Framework. RFC 2330, 1998.

[6] Traffic Monitoring using sFlow. http://www.sflow.org/sFlowOverview.pdf.

[7] Netflow. http://www.CISCO.com/warp/public/732/Tech/nmp/netflow/index.shtml.

[8] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, 2003.

[9] Cristian Estan, et al. Building a better NetFlow. ACM SIGCOMM Computer Communication Review, 2004,34(4).

[10] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422～426.

[11] Cristian Estan, and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Transactions on Computer Systems (TOCS), 2003, 21(3): 270～313.

[12] Kumar Abhishek, Jun Xu, and Jia Wang. Space-code bloom filter for efficient per-flow traffic measurement. Selected Areas in Communications, 2006, 24(12): 2327～2339.

[13] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, 2003.

[14] Sriram Ramabhadran, and George Varghese. Efficient implementation of a statistics counter architecture. ACM SIGMETRICS Performance Evaluation Review, 2003, 31(1).

[15] Thomas M. Chen, and J-M. Robert. Worm epidemics in high-speed networks. Computer, 2004, 37(6): 48～53.

[16] Jingyu Qiu, and Edward W. Knightly. Measurement-based admission control with aggregate traffic envelopes. IEEE/ACM Transactions on Networking (TON), 2001, 9(2): 199～210.

[17] Haoyu Song, et al. Fast hash table lookup using extended bloom filter: an aid to network processing. ACM SIGCOMM Computer Communication Review, 2005, 35(4).

[18] George Bakos, and Vincent H. Berk. Early detection of internet worm activity by metering ICMP destination unreachable messages. International Society for Optics and Photonics, 2002.

[19] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS),2003, 28(1): 51～55.
[20] Baek-Young Choi, Jaesung Park, and Zhi-Li Zhang. Adaptive random sampling for load change detection. ACM SIGMETRICS Performance Evaluation Review, 2002,30(1).
[21] David Mooreet al. Internet quarantine: Requirements for containing self-propagating code. 22nd Annual Joint Conference of the IEEE Computer and Communications, 2003, 3.
[22] OC48 trace. https://datacaida. org/datasets/oc48/oc48-original/. Trace file: 20030424-000000-0-anon. pcap. gz.
[23] Juniper Networks. T Series Core Routers Architecture Overview. http://www. juniper. net/us/en/local/pdf/whitepapers/2000302-en. pdf.
[24] Juniper Networks. Securing Data Centers with the SRX Series Services Gateways . http://www. juniper. net/us/en/local/pdf/appnotes/3500138-en. pdf.
[25] Internet World Stats. INTERNET USAGE STATISTICS 2011. http://www. internetworldstats. com/stats. htm.
[26] CNNIC. 第 29 次中国互联网络发展状况统计报告. http://www. cnnic. cn/research/bgxz/tjbg/201201/t20120116_23668. html.
[27] 林闯,单志广,任丰原. 计算机网络的服务质量(QoS). 北京:清华大学出版社,2004.
[28] Kenjiro Cho. Managing Traffic with ALTQ. USENIX Annual Technical Conference, FREENIX Track, 1999.
[29] Werner Almesberger. Linux network traffic control—implementation overview, 1999.
[30] Main Page of Untangle. http://www. untangle. com/.
[31] Main Page of M0n0wall. http://m0n0. ch/wall/.
[32] Main Page of pfSense . http://www. pfsense. org/.
[33] Main Page of Panabit Limited. http://www. panabit. com/.
[34] Main Page of KuanGuang Telecom. http://www. kuanguang. com. cn/.
[35] Main Page of QQTechnology. http://www. qqtechnology. com/.
[36] Nabil Benameur, et al. Quality of service and flow level admission control in the internet. Computer Networks, 2002, 40(1): 57～71.
[37] Srisankar Kunniyur, and Rayadurgam Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. ACM SIGCOMM Computer Communication Review,2001, 31(4).
[38] Thomas Bonald, and James Roberts. Scheduling network traffic. ACM SIGMETRICS Performance Evaluation Review, 2007, 34(4): 29～35.
[39] Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. ACM SIGCOMM, 1999.
[40] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A stochastic model of TCP/IP with

stationary random losses. ACM SIGCOMM Computer Communication Review，2000，30(4)：231～242.

[41] Abhay K. Parekh，and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks：the single-node case. IEEE/ACM Transactions on Networking (TON)，1993，1(3)：344～357.

[42] Madhavapeddi Shreedhar，and George Varghese. Efficient fair queuing using deficit round-robin. IEEE/ACM Transactions on Networking，1996，4(3)：375～385.

[43] Myung-Sup Kim，Young J. Won，and James Won-Ki Hong. Application-level traffic monitoring and an analysis on IP networks. ETRI journal，2005，27(1)：22～42.

[44] Internet Assigned Numbers Authority. http://www.iana.org/.

[45] Hun-Jeong Kang，Myung-Sup Kim，and James Won-Ki Hong. A method on multimedia service traffic monitoring and analysis. Self-Managing Distributed Systems，2003.

[46] Jacobus Van Der Merwe，Ramon Caceres，Yang-hua Chu，and Cormac Sreenan. Mmdump：A tool for monitoring Internet multimedia traffic. ACM SIGCOMM Computer Communication Review，2000，30(5)：48～59.

[47] Subhabrata Sen，Oliver Spatscheck，and Dongmei Wang. Accurate，scalable in-network identification of p2p traffic using application signatures. In Proceedings of the 13th international conference on World Wide Web，2004.

[48] Sebastian Zander，Thuy Nguyen，and Grenville Armitage. Self-learning IP traffic classification based on statistical flow characteristics. Passive and Active Network Measurement，2005.

[49] Denis Zuev，and Andrew W. Moore. Traffic classification using a statistical approach. Passive and Active Network Measurement，2005.

[50] Anthony McGregor，et al. Flow clustering using machine learning techniques. Passive and Active Network Measurement，2004.

[51] Sally Floyd，and Van Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking，1993，1(4)：397～413.

[52] P2P packet classifier. http://www.ipp2p.org/.

[53] OpenDPI. The Open Source Deep packet Inspection Engine. http://www.opendpi.org/.

[54] L7-filter. Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/.

[55] Reference on CBQ. http://www.icir.org/floyd/cbq.html.

[56] 李国栋. 高速网络流量管理系统的设计与实现. 北京：清华大学出版社，2010.

[57] 陈震等. TCG-TNC 可信网络连接系统设计与实现. 信息网络安全，2009，(07)：49～52.

[58] 韩阜业等. 基于覆盖网的协同式网络安全防护与分析系统. 信息网络安全，2012，(04)：7～13.

[59] 安德鲁·斯图尔特·塔能鲍姆. 计算机网络(第 4 版). 潘爱民译. 北京：清华大学出版社，2004.

[60] 巢剑. IP/ATM 队列调度算法的理论研究与性能分析. 硕士学位论文. 成都：成都电子科技大学，2002.

[61] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for broadband applications. 13th

Networking for Global Communications，1994.

[62] Sriram Ramabhadran，and Joseph Pasquale. The stratified round robin scheduler：design，analysis and implementation. IEEE/ACM Transactions on Networking （TON），2006，14（6）：1362～1373.

[63] Jon CR Bennett，and Hui Zhang. WF2Q：worst-case fair weighted fair queueing. 5th Annual Joint Conference of the IEEE Computer Societies，1996.

[64] Hideyuki Shimonishiet al. An improvement of weighted round robin cell scheduling in ATM networks. Global Telecommunications Conference，1997.

[65] Madhavapeddi Shreedhar，and George Varghese. Efficient fair queueing using deficit round robin. ACM SIGCOMM Computer Communication Review，1995，25(4)：231～242.

[66] Guo Chuanxiong. SRR：An O（1）time complexity packet scheduler for flows in multi-service packet networks. ACM SIGCOMM Computer Communication Review，2001，31(4).

[67] Shun Yan Cheung，and Corneliu S. Pencea. BSFQ：bin sort fair queueing. INFOCOM，2002.

[68] Bogdan Caprita，Jason Nieh，and Wong Chun Chan. Group round robin：improving the fairness and complexity of packet scheduling. In Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems，2005.

[69] Abhay K. Parekh，and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks：the single-node case. IEEE/ACM Transactions on Networking（TON），1993，1(3)：344～357.

[70] Jon CR Bennett，and Hui Zhang. Hierarchical packet fair queueing algorithms. IEEE/ACM Transactions on Networking，1997，5(5)：675～689.

[71] Fabio Checconi，P. Valente，and L. Rizzo. QFQ：Efficient Packet Scheduling with Tight Bandwidth Distribution Guarantees. IEEE/ACM Transactions on Networking，2013.

[72] L. Lenzini，E. Mingozzi，and G. Stea. Aliquem：a novel DRR implementation to achieve better latency and fairness at O(1) complexity. 10th IEEE International Workshop on Quality of Service，2002.

[73] Rahul Garg，and Xiaoqiang Chen. RRR：Recursive round robin scheduler. Computer Networks，1999，31(18)：1951-1966.

[74] Chuanxiong Guo. G-3：An O(1) time complexity packet scheduler that provides bounded end-to-end delay. 26th IEEE International Conference on Computer Communications，2007.

[75] Pawan Goyal，Harrick M. Vin，and Haichen Chen. Start-time fair queueing：a scheduling algorithm for integrated services packet switching networks. ACM SIGCOMM Computer Communication Review，1996，26(4).

[76] Hanrijanto Sariowan，Rene L. Cruz，and George C. Polyzos. Scheduling for quality of service guarantees via service curves. 4th International Conference on Computer Communications and Networks，1995.

[77] Rene L. Cruz. Service burstiness and dynamic burstiness measures：A framework. Journal of

High Speed Networks，1992，1(2)：105～127.

[78] Rene L. Cruz. Quality of service guarantees in virtual circuit switched networks. IEEE Journal on Selected Areas in Communications，1995，13(6)：1048～1056.

[79] Ion Stoica，Scott Shenker，and Hui Zhang. Core-stateless fair queueing：a scalable architecture to approximate fair bandwidth allocations in high-speed networks. IEEE/ACM Transactions on Networking (TON)，2003，11(1)：33～46.

[80] Constantinos Dovrolis，Dimitrios Stiliadis，and Parameswaran Ramanathan. Proportional differentiated services：Delay differentiation and packet scheduling. ACM SIGCOMM Computer Communication Review，1999，29(4)：109～120.

[81] Constantinos Dovrolis，and Parameswaran Ramanathan. A case for relative differentiated services and the proportional differentiation model. Network，1999，13(5)：26～34.

[82] H. El-Gebalyet al. Performance evaluation of a window-based approach for ATM cell scheduling. Canadian Conference on Electrical and Computer Engineering，1995.

[83] 韩阜业.基于覆盖网的协同式僵尸网络压制系统的设计与实现，校级优秀硕士学位论文，北京：清华大学，2011.

第3章

互联网流量档案化

随着互联网应用的普及和渗透,宽带移动无线网络的大规模商用,整个互联网流量总量一直保持着高速增长,思科公司报告[1]预言,互联网的流量数据在2011到2016年之间将增长4倍,并于2016年达到1.3ZB。即使对于一家大型的互联网公司,在日常运营中生成和累积的用户行为数据是相当庞大的,往往达到TB级甚至PB级的数据。如此庞大的数据流量进行管理是一个具有挑战性的问题。在通用多核处理器平台上,为了适应线速获取流量数据,本章对网包处理过程中的相关基础知识和相应技术手段进行基本介绍。其中的核心关键技术是万兆级数据包从网卡到内存的复制机制和应用层网络协议的软件并行化方案。

3.1 高速网包获取的关键技术

3.1.1 网包

随着互联网技术的发展,网络安全问题日益凸显,由于其开放性,使得网络要遭受大量可知或未知的攻击。在这种背景下出现很多网络安全防范技术,例如,入侵检测系统、特征码检测和安全扫描技术等,但是很多攻击是无法进行及时检测和预防的,需要通过对网络网包的捕获来实现对数据信息的收集,便于以后的分析和使用。

网络之间在传递信息时,单个信息被划分为多个数据块,并以其作为传输单位进行发送,每个小块可能会沿着不同的路径在一个或多个网络中传输,并在目的地实施重组,这些小块就是“网包”。TCP/IP协议簇中将这些网包根据其包含信息不同被分为网络节点端口网包、IP网包、传输层网包和应用层网包。

3.1.2 Linux-NAPI

每当接收到一个链路层以太网帧时,设备驱动都会为了及时处理到来的网包而产生

一个中断事件，可是在高流量负载的情况下，CPU 会花费大量的时间在中断处理上，从而造成 CPU 资源的浪费。而 NAPI 将中断和轮询相结合，取代以往的中断处理模型。由于中断处理变少，CPU 的负载被大大降低，并且通过轮询的方式使得各设备之间的处理变得更加公平。

当某个设备刚接收到新帧时，依旧使用中断通知系统，并将该设备注册到设备轮询队列中，关闭对该设备的中断响应。轮询队列中注册的网络设备进行轮询，从中读取网包。采用配额的方法保证对各个网络设备的公平调度。如果配额不足以处理缓冲区的数据，则将该设备移至轮询队列的尾端，如果配额足够处理缓冲区的数据，则将该设备从轮询队列中注销，同时打开该设备的中断响应，处理过程如图 3.1 所示。

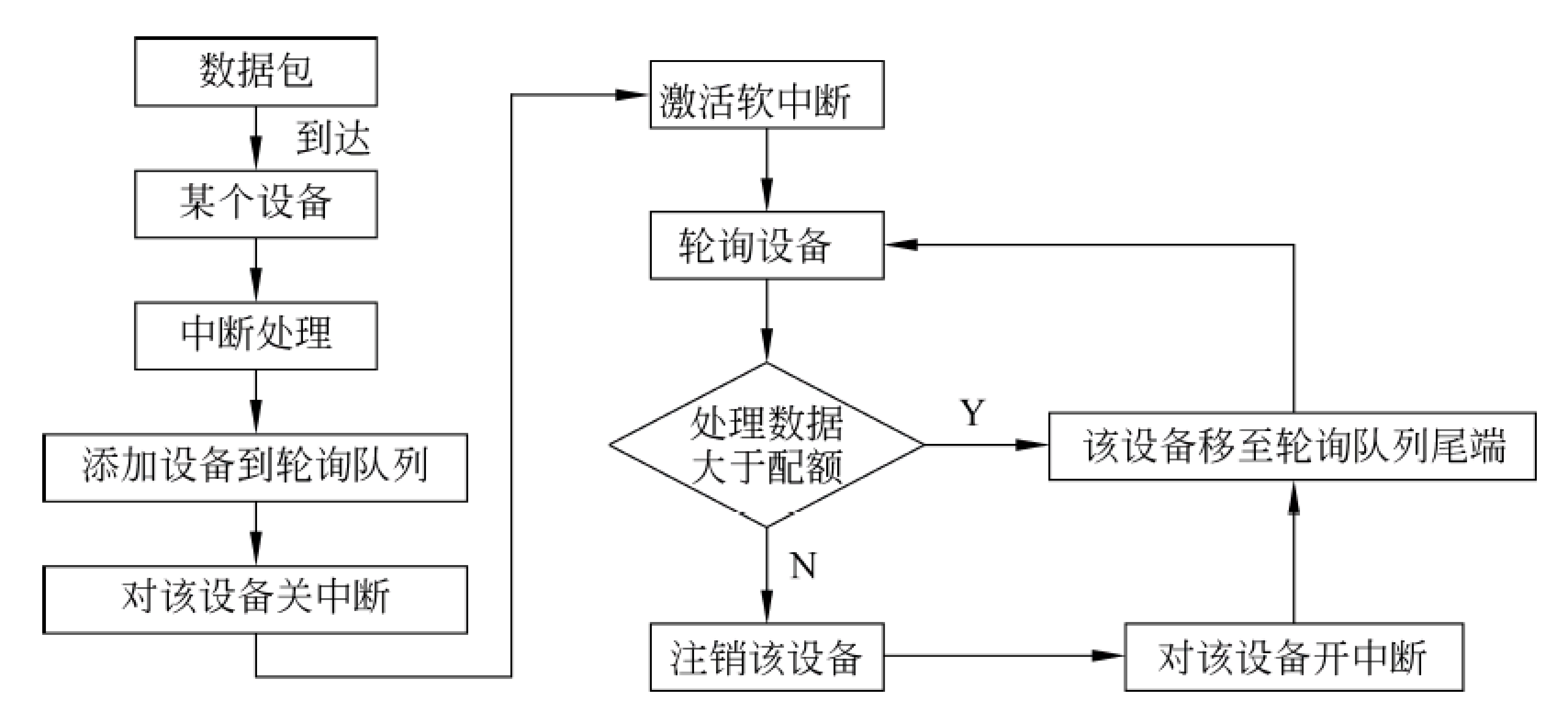


图 3.1　NAPI 处理过程

虽然 NAPI 能够减少 CPU 的资源浪费，但是只有很少的设备使用 NAPI，因为它存在一些比较严重的缺陷，例如，系统不能及时地处理每次接收的网包，而且随着传输速度增加，累计的网包将会耗费大量的内存等，所以 NAPI 只适应处理高速率短数据包的情况。

3.1.3　libpcap

目前有两种方法从网络中捕获网包，一种是利用专用的硬件，这种方法虽然性能较好，但是昂贵的价格和较差的扩展性让使用者望而却步；另一种是利用软件来实施数据流量的捕获，虽然较之专用硬件性能较弱，但其低廉的成本和强扩展性得到了人们的广泛青睐。下面将介绍一个包捕获和解析的开源库——libpcap。

libpcap[2] 是一个平台独立的 API 接口函数库，能够独立访问链路层，并读取数据，可用于 user-level 的网包捕获工作。libpcap 是一个高层的编程接口，使得程序的编写和移植工作变得方便简单。很多网络安全产品都以它为基础，在数据链路层上实现网络数据的捕获和分析。libpcap 使用了 BPF 过滤机制，可以过滤掉不需要的网包，而只捕获用户

感兴趣的网包。libpcap 也可以将网络上捕获的网包存储到特定的文件格式中，并能随时从该文件中读取网包的信息。

libpcap 库提供的主要的功能函数如下。

（1）pcap_lookupdev()：扫描有效网络设备端口，返回供 pcap_open_live()调用的设备名指针。

（2）pcap_lookupnet()：获得指定设备的网络地址及网络掩码。

（3）pcap_open_live()：打开指定网络设备的接口用于捕获网包。

（4）pcap_compile()：过滤字符串编译到过滤程序中，例如上述的 BPF 过滤规则。

（5）pcap_setfilter()：设置网络过滤器。

（6）pcap_dispatch()：捕获网包进行中，后调用 recvfrom()把网卡捕获的网包存储到用户指定的空间。

（7）pcap_loop()：用于循环捕获网包。

（8）pcap_close()：网包捕获结束，释放资源。

libpcap 捕获函数库的工作过程如图 3.2 所示。

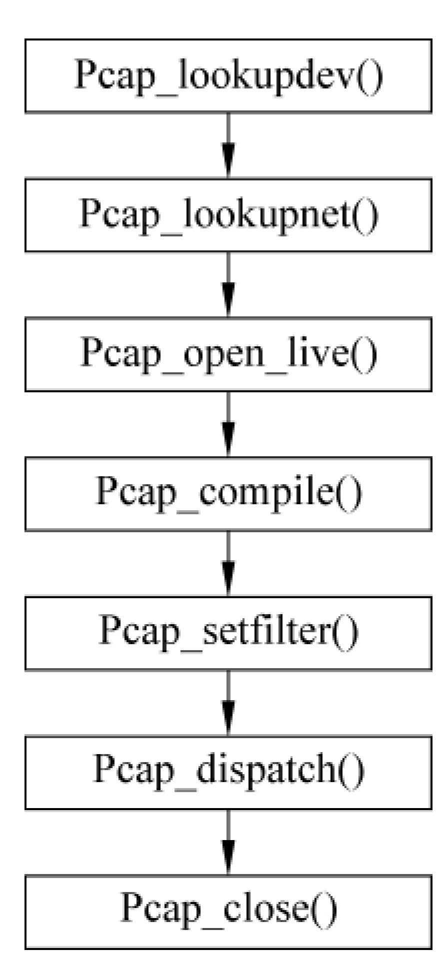


图 3.2　libpcap 捕获函数库的工作流程

packet reader thread 从一个网络接口那里捕获包，并根据过滤规则的设置丢弃一些包。Buffer 中含有预先填好的 pcap file 头部，如图 3.3 所示，捕获的包和相应的包头部一起复制到 buffer 中，如果一个 buffer 填满了，就填写下一个。packet reader 把包写在一个能立即转储到磁盘上的格式。写 buffer 到 disk 上是使用第二个线程，以保证快速转储。

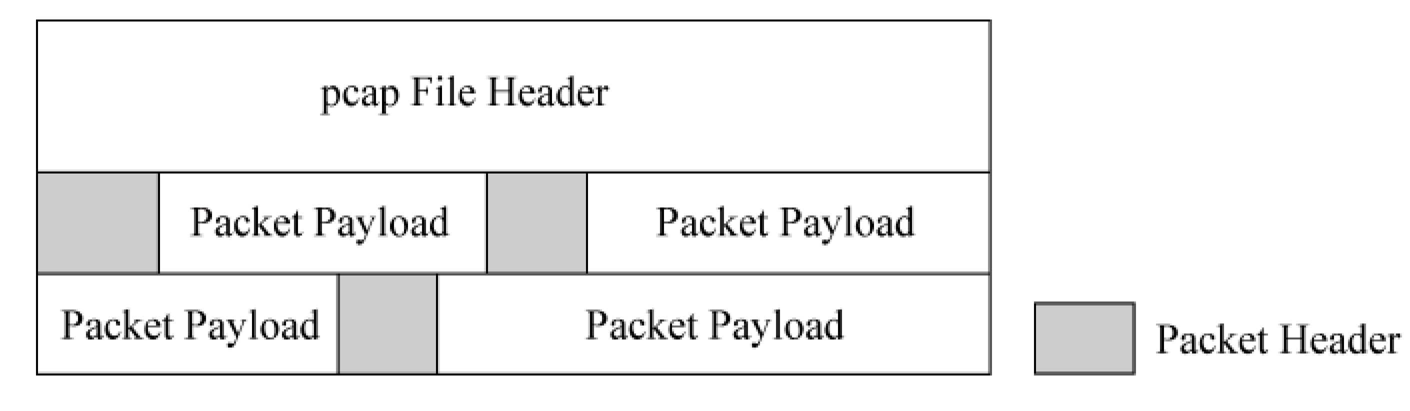


图 3.3　pcap 文件格式

3.1.4　PF_RING

利用 libpcap 捕获流量时，CPU 将花费大量的时间将网卡上获得的网包经过内核函数处理发送到用户空间，整个过程是从网卡到内核，再到用户空间，大大降低了 CPU 的处理效率。

Luca Deri 等[2]提出了 PF_RING 的技术，PF_RING 将网卡接收的网包存储在一个环状的缓存中，这个环状缓存有两个接口，一个可供网卡向其中写入网包，另一个则是供应用层提供读取网包的接口，读取网包的接口可以通过 mmap 实现，mmap()可以建立核心态内存与用户内存空间的映射，函数最后返回值是用户态内存的首地址 pA。这样，可以通过调用 recvfrom()将捕获的网包直接从网卡上传送到 pA 为首地址的用户态内存，减少数据的复制及其他处理操作，减少 CPU 的处理时间，某种程度上，提高了网包捕获的效率。

PF_RING 结构如图 3.4 所示。

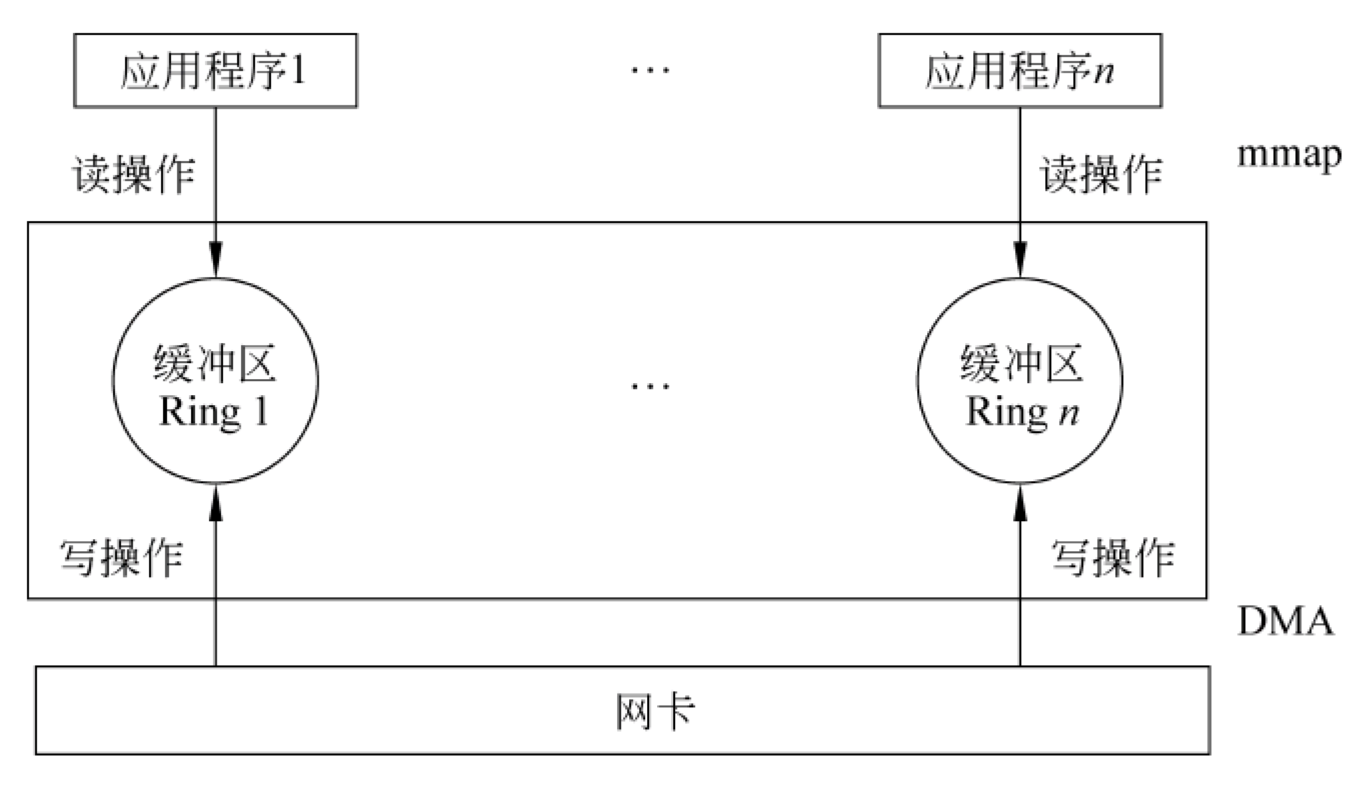


图 3.4 PF_RING 结构

3.1.5 Netmap

R. Lizzo 等[3]提出了 Netmap 框架，如图 3.5 所示。Netmap 是高性能的应用层收发包框架，主要解决收发包过程中产生巨大开销的 3 个方面的问题，分别是每个包的动态内存分配的开销、系统调用的开销和内存副本的开销。

Netmap 主要是通过批处理的方式减少系统调用的次数，通过内核和用户态共享缓冲区，即类似 3.1.4 节介绍的 PF-RING 技术，以避免内核态和用户态的两次复制，并通过预分配固定大小的 buff 来保存网包代替原本的动态分配来解决的。

通过实验数据验证 Netmap 的收发包框架性能测试比一般标准的 APIs 至少快一个数量级[3]，并且提供的框架是和现有的 OS 系统紧密相连的，不需要特殊硬件的支持，使用简单，易于维持，重要的是 Netmap 并不是以牺牲安全为代价提升性能，

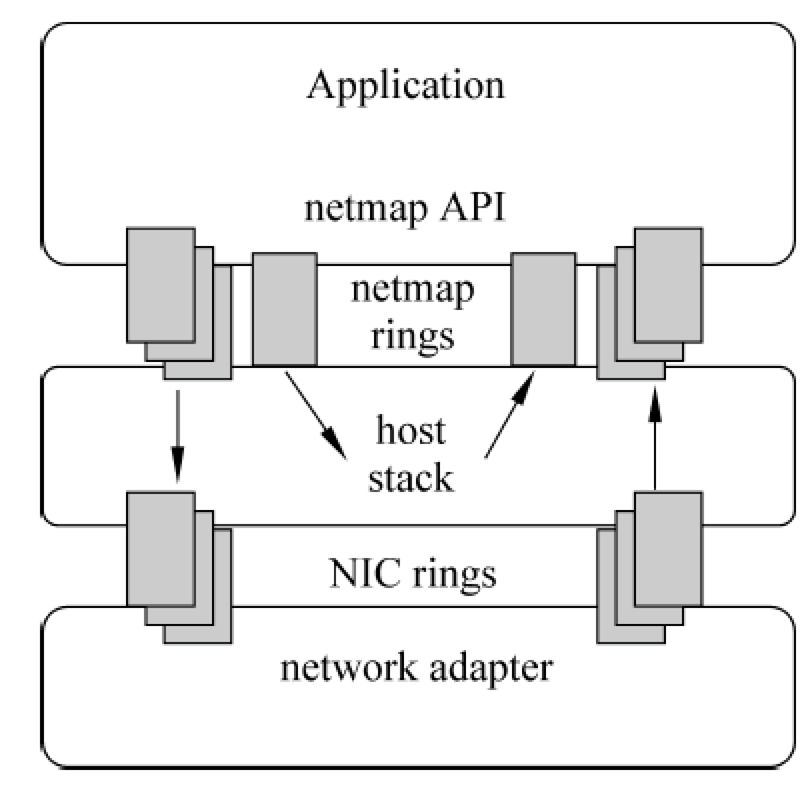


图 3.5 Netmap 框架示意图

Netmap 可以确保内核永远不崩溃。

3.1.6　Scap

很多网络监测应用需要分析网络层之上面向连接的流量，以及在 TCP 分段基础上实现的弹性规避尝试的流量。但是，现有的网络流量采集架构，仅提供对原始网包的抓取，需要安全应用的开发人员后续复杂的操作，例如，对网流的追踪和 TCP 流的重组。这种差距导致应用程序的复杂性增加，较长的开发时间，以及最重要的是，由于在网包捕获的子系统和流处理的模块中产生了过多的数据备份，导致处理性能的降低。

网流捕获库(Scap)基于一个内核模块，直接处理流跟踪和 TCP 流重组，Scap 交付给用户层应用的是流级(Flow Level)统计，最大限度地减少数据移动操作，丢弃在早期阶段不感兴趣的需要重组的流量。Scap 本身支持在多核架构上并行处理，采用了网卡的最新功能。

Scap 架构如图 3.6 所示。

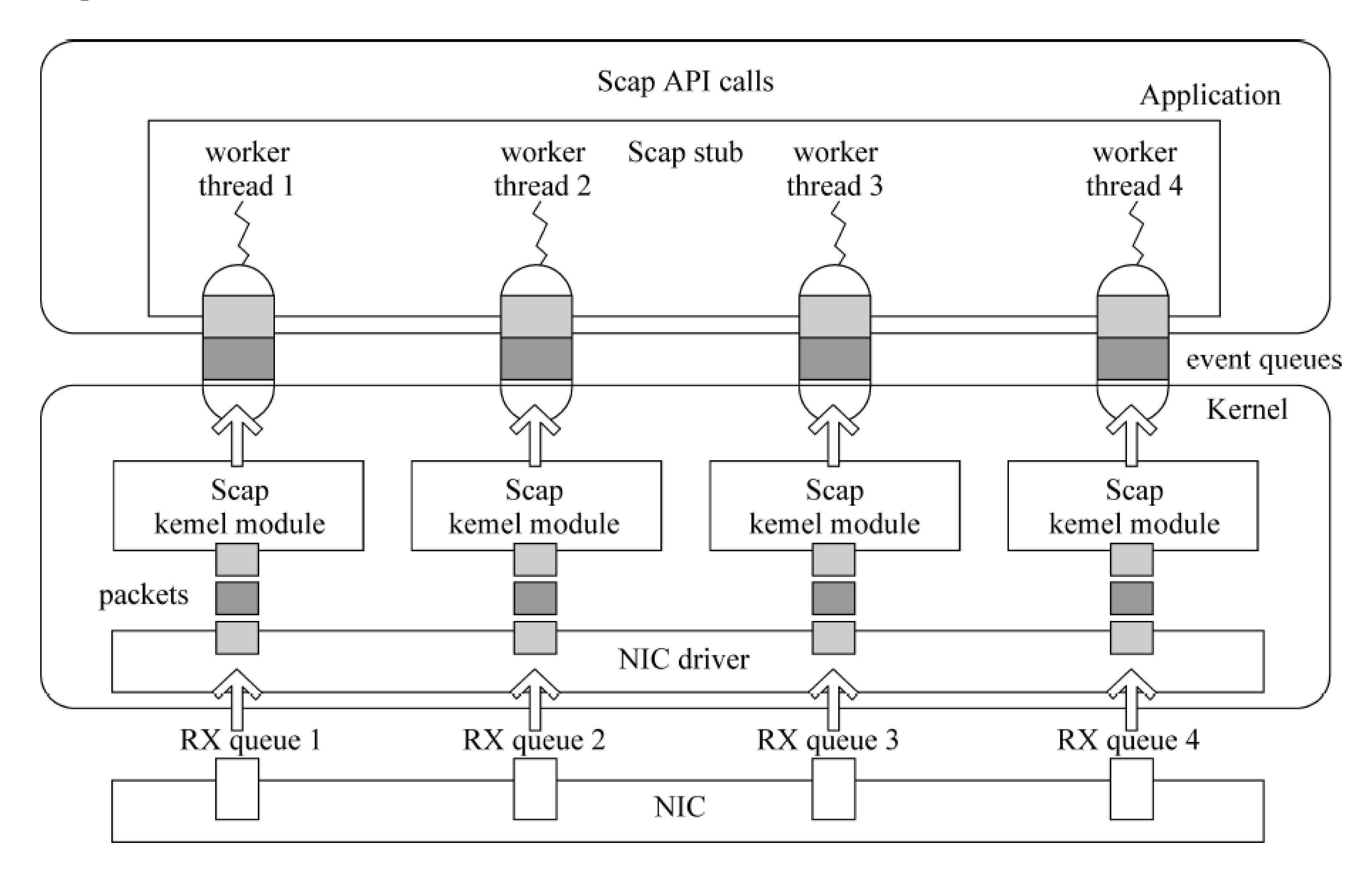


图 3.6　Scap 架构示意图

其中使用的关键技术概括如下。

(1) 在内核里增加可在 10ms 内增删的 Flow Director filters(FDIR)，根据网包的信息(目标 IP、源端口、目标端口和协议名称等)对流里面的网包进行分队，对无用的分队进行早期丢弃。

(2) 引入 Prioritized Packet Loss(PPL)技术，预先对不同的流进行优先级标记，并针

对不同优先级的流设置不同的内存阈值，当内存超出某个优先级的阈值时丢弃不高于该优先级的流，这样就代替了原有随机丢弃的方式，避免了重要信息的遗漏。

（3）对不同的流进行不同的重组方式。

（4）对于同一个流的操作在同一个核处理（通过建立 kernel 和 worker thread），并且针对多核情况，利用动态规划来平均分配每个核的工作量。

（5）对每个流设置一个截短大小（cutoff size），只保留截短大小（cutoff size）以内的网包（以及流的头和尾），剩下的丢弃。

（6）为了平衡多网卡队列和核上的网络流量负载，Scap 使用静态的 hash-based 方法（例如 RSS（Receive Side Scaling）和动态负载平衡方法。

Scap 架构由于事先进行合理的预处理，导致丢包率有明显的改善，而且在网络拥堵的情况下能够优先照顾用户需要的包。

上一小节介绍的 netmap 也是对包捕获的改进，为此，表 3.1 是将 Scap 和 Netmap 的关键技术进行了简单对比。

表 3.1　Scap 和 Netmap 的比较

使用技术	Scap	Netmap
处理粒度	Stream	Packet
Packet copy	Subzero	Zero
过载丢弃策略	基于流的优先级	随意丢弃

3.2　网包位图索引压缩算法

3.2.1　位图索引数据库

传统的关系型数据库是面向更改的，存储在数据库中的数据需要经常改动。而位图索引数据库专门为科学数据设计，这些数据通常是由科学仪器或是科学仿真产生的，特点是数据量极其大，而且不再更改。位图索引数据库解决了如何在海量的科学数据中快速地找出那些少量被需要的数据的问题，而传统关系型数据库并不适合这项任务。

位图索引数据库中用到的技术主要是位图索引、位图压缩和归类。在位图索引数据库中，数据是按列存储的，一个列的数据存储在一起，并做位图索引。一个简单的位图索引的例子如表 3.2 所示。其中 RowID 表示对应值在表中第几行，生成的索引是一个矩阵，矩阵中每一行只有一个 1，其余都是 0，标 1 的位置对应于该行数据在这一列上的取值。这样生成的位图索引有一个比较大的缺点，索引的列数随着取值的多样性而线性增长。为了控制索引的大小和查询时间，需要对索引压缩和归类。压缩是减小索引中大量

0 带来的空间消耗，归类是对位图索引的一些列的合并。比如值 1.01 和 1.02 可以归类成 1。通过归类可以减少位图索引的列数，增加查询和储存的效率。

表 3.2　位图索引示例

RowID	列 Number	Bitmap 索引			
		=1	=2	=3	=4
1	1	1	0	0	0
2	2	0	1	0	0
3	2	0	1	0	0
4	3	0	0	1	0
5	4	0	0	0	1
6	1	1	0	0	0

我们将主要介绍 3 种位图索引压缩算法：WAH[9]、PLWAH[10] 和 COMPAX[11]。其中 PLWAH 和 COMPAX 都有改进版，PLWAH 的改进版为 PLWAH with adaptive counter，原文没有明确写出 PLWAH 本身是否带 adaptive counter，因此此处当做改进版，而 COMPAX 的改进版是 COMPAX2（双边）和 COMPAX+oLSH（重排序）。

3.2.2　WAH 索引压缩算法

WAH 索引压缩算法示例如图 3.7 所示。

128b	1*1,20*0,3*1,79*0,25*1			
31-b groups literal(hex) WAH(hex)	1,20*0,3*1,7*0 40000380 40000380	62*0 00000000 00000000 80000002	10*0,21*1 001FFFFF 001FFFFF	4*1 0000000F 0000000F

图 3.7　WAH 索引压缩算法示例

WAH 算法是 FastBit 位索引数据库的默认算法，将位序列分成以 31b（对于 WAH64 就是 63b）为单位的 group。这样，group 有两种类型。

（1）Literal，即这 31b 中有 0 有 1。

（2）Fill，即这 31b 全为 0 或者全为 1。

在 WAH 算法中，最终形成的 word 是 32b，其中最高位为类型的标志位。Literal 类型的 Group：类型标志位为 0，余下的 31b 即为原来的 literal group；Fill 类型的 Group：分为 1-Fill 和 0-Fill。此时 32b 中类型标志位为 1，第二位作为 Fill 类型的标志（0-Fill 即为 0，1-Fill 即为 1），余下为 30b 作为 counter，表示连续出现多少个 0-Fill（或者 1-Fill）的

group。

3.2.3 PLWAH 算法

PLWAH 算法和 WLAH 类似，同样是将原始码流分成以 31b 为单位的 group。Group 也和 3.2.2 节介绍的相同，一样分成 Literal 和 Fill 两种类型，但是压缩方式有变化，具体如下所示。

第一步：依照 WAH 方法对于 Fill-group 和 Literal-Group 添加标志位与编码，形成一组以 32b word 为单位的编码（标志位为 0 称为 literal-word，标志位为 1 的称为 fill-word，下同）。此处的区别是对于 fill-word 只有低 25b 作为 counter（WAH 方法是低 30b 都是 counter，PLWAH 要留出中间的 5b 作为 position list，下面会用到）。

第二步：检查每个 fill-word 后的 word。如果下一个 word 是 literal-word 且是 nearly identical（nearly identical 定义是 literal-word 和上一个 fill-word 的差异小于等于 s 位，s 此处暂时为 1，后面会进一步讨论），则在 fill-word 的 position list 上填入下一个 word（此时为 literal-word）的差异位位置（此处是 31b，因此差异位标号为 1～31，留出 5b 目的在此），同时删去下一个 word（因为信息已经保存在此 fill-word 中，没有必要继续留着），若 fill-word 后的 word 是如下 3 种情况。

（1）异类型的 fill-word。

（2）非 nearly-identical 的 literal-word。

（3）同类型的 fill-word（这种情况产生的原因是连续的 fill-group 超出了 1 个 fill-word 的 counter 的计数范围），则 position list 不变。

进行扩展讨论：

对于 32 位 PLWAH，由于 nearly identical 的定义中的 s 不为 1，则在第一步（即利用 WAH 原理编码）时需要留出 $5\times s$ 位 position list，余下的 $32-2-5\times s$ 位为 counter。在第二步时，向 fill-word 中的 position list 添加差异位时须按序添加差异位位置（5b 标识一个差异位，$5\times s$ 位标识至多 s 个差异位），当 $s=0$ 时 PLWAH 退化成 WAH。

对于 64 位 PLWAH，position list 以 6 为单位，即留出 $6\times s$ 位为 position list，余下的 $64-2-6\times s$ 位为 counter。

PLWAH＋adaptive counter（以 PLWAH32，$s=1$ 为例）：

对 PLWAH 进行了小幅改进，即考虑连续出现极多的 0 或者 1 以至于 1 个 word 的 counter 无法完全容纳。此处选用的方法为使用两个连续的同类型 fill-word，把两个 counter 部分视为一个大的 counter，第一个 fill-word 记录 LSB25 位，第二个 fill-word 记录 MSB25 位（相当于 50b 的大 counter），同时第一个 fill-word 的 position list 为空，第二个 fill-word 的 position list 照常进行计算，如图 3.8 所示。这样的好处是无须扩展 word 位数即可完成对更多连续码流的编码任务，节约 index 空间。

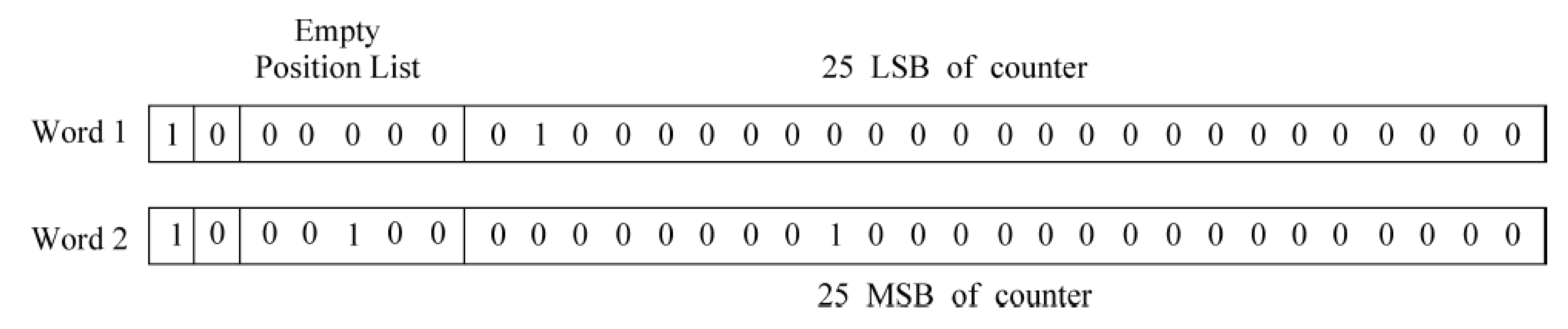


图 3.8 PLWAH 改进举例

3.2.4 COMPAX 算法

COMPAX 压缩算法示例如图 3.9 所示。

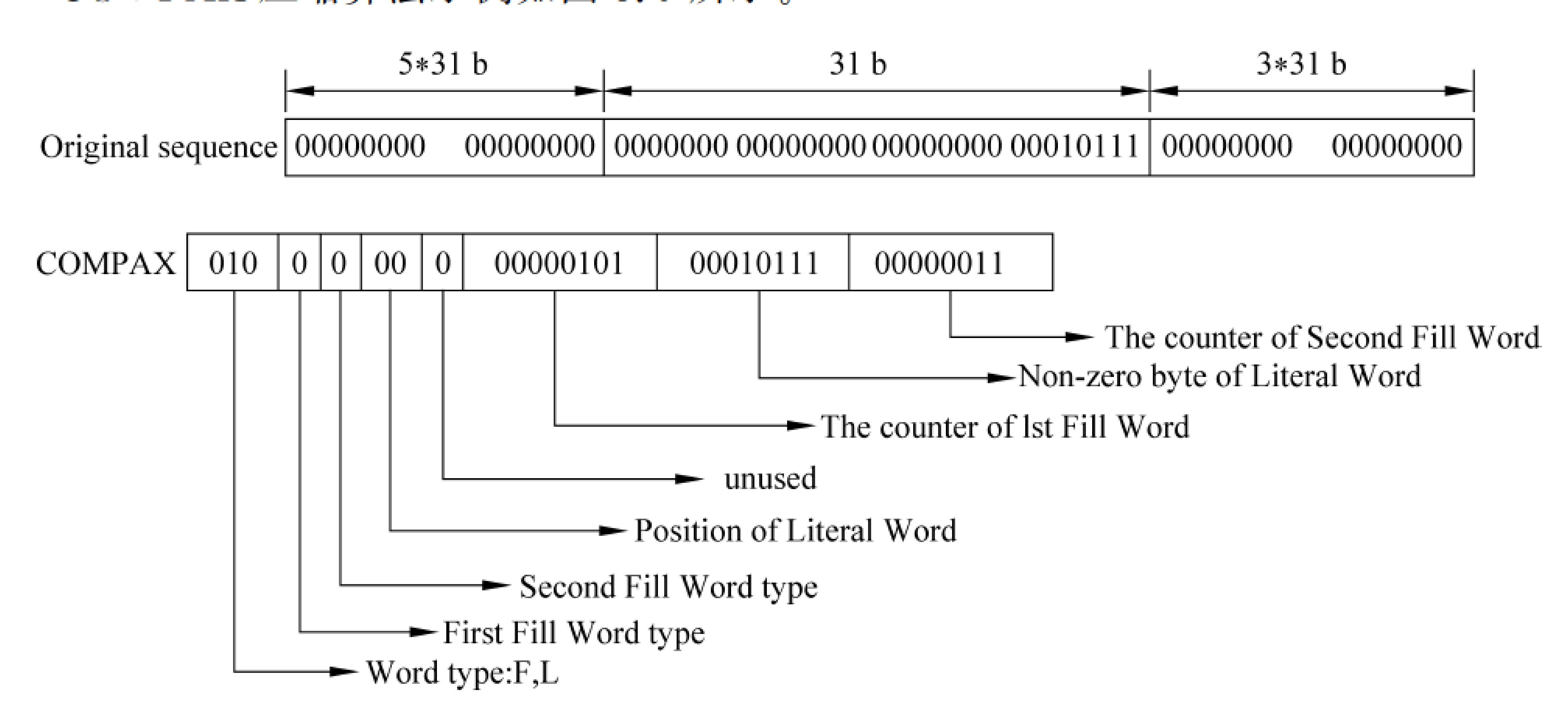


图 3.9 COMPAX 压缩算法示例

COMPAX 算法也是 WAH 的改进，这里以 32b 为例，但 COMPAX 的标志位相对较多。这里 Literal 和 Fill 的定义同 WAH 和 PLWAH。同样是每 31b 分成一个 group，并且将这些 group 按照以下特征分组。

（1）Literal-Fill-Literal（LFL），即 1 个 literal group＋N 个 Fill group＋1 个 literal group，且这两个 literal group 的非 0 位（或者非 1 位）在同一个 byte 上（一个 group 在前面补一位 0 即构成 4 个完整的 byte，要求非零位在同一个完整的 byte 上）。

（2）Fill-Literal-Fill（FLF），即 N 个 Fill-group＋1 个 literal group＋N 个 fill-group（对 literal group 的要求同上）。

（3）Fill（F），分为 0-Fill 和 1-Fill，无法按照（1）和（2）进行分组的 fill-group 即归入此类型。

（4）Literal（L），无法按照（1）和（2）进行分组的 literal group 即归入此类型。

对于这 4 种类型，就有 4 种不同的 word。

(1) L-word 第一位为标志位 1,余下 31b 即为原来的 literal group。

(2) F-word 第一位为标志位 0,对于 0-Fill 第二、三位为 00,对于 1-Fill 第二、三位为 11。余下 29b 为 counter,即记录有多少个连续这样的 group。

(3) LFL-word 第一位为标志位 0,第二、第三位为 01,第 4、5 位(2b)标识第一个 literal group 的非零字节位置(共 4B,标号为 0～3),第 6、7 位(2b)标识第二个 literal group 的非零字节位置(共 4B,标号为 0～3)第 8 位标识 F 类型,0 为 0-Fill,1 为 1-Fill;第 9～16 位(8b,1b)为第一个 literal group 中非零字节,第 17～24 位(8B,1B)为 Fill 的 counter,标示有多少个连续的 fill group(即多少个连续 31b 的 0/1),第 25～32 位(8b,1B)为第二个 literal group 中的非零字节。

(4) FLF-word 第一位为标志位 0,第二、三位为 10,第 4、5 位为第一个 fill 的类型(0-Fill 为 00,1-Fill 为 11,下同),第 6、7 位为第二个 fill 的类型,第 8 位空闲。第 9～16 位(8b)为第一个 fill 的 counter,第 17～24 位(8b,1B)为 literal group 的非零字节,第 25～32 位为第二个 fill 的 counter。

在已知 COMPAX 的情况下,具体读码方式如下。

(1) 如果第一位为 1,为 L-word。

(2) 如果第一位为 0:

① 第二、三位为 00:0-fill word。

② 第二、三位为 11:1-fill word。

③ 第二、三位为 01:LFL。

④ 第二、三位为 10:FLF。

3.3 流量归档查询系统

3.3.1 基于关系数据库的系统实现

1. 关系型数据库

关系型数据库是目前在软件设计与实现中使用最广的数据库类型。关系型数据库是指基于关系模型设计的数据库,通过数据集内部共有特征将数据联系在一起。在关系型数据库中,有相似结构的数据存放在同一个关系中,这种关系也称为表。每个表中包含一些列,每一列是描述该关系的一个属性,而每一行都具有每一列的属性,行与行之间相互独立。关系型数据库通常是存储金融信息、医疗信息、个人信息、制造信息和后勤信息的首选。常用的商业关系型数据库有 Oracle 和 Microsoft SQL Server,开源的有 MySQL 和 SQLite 等。

2. 基于 MySQL 数据库的系统实现

本节中采用的关系型数据库是使用频率最高的 MySQL 数据库。

根据索引的定义，MySQL 数据库中存放索引表的结构如表 3.3 所示。对某一列的数据是否建索引将影响着有关这一列的查询速率。建立列索引会明显的增加一条记录插入的耗时。因此在选择是否建索引时，应该选择那些数据重复率很小(基本不重复)，并且在查询时最常使用到的那些列。在这些列中，有该特点的列只有时间戳。本节中将讨论对所有列都不建索引和仅对时间戳列建索引两种情况下的系统的性能变化。

表 3.3　MySQL 数据中存放索引的表的结构

索引设计		MySQL 中表的设计		
域	类　型	列　名	类　　型	是否列索引
sip1	byte	sip1	TINYINT UNSIGNED NOT NULL	否
sip2	byte	sip2	TINYINT UNSIGNED NOT NULL	否
sip3	byte	sip3	TINYINT UNSIGNED NOT NULL	否
sip4	byte	sip4	TINYINT UNSIGNED NOT NULL	否
dip1	byte	dip1	TINYINT UNSIGNED NOT NULL	否
dip2	byte	dip2	TINYINT UNSIGNED NOT NULL	否
dip3	byte	dip3	TINYINT UNSIGNED NOT NULL	否
dip4	byte	dip4	TINYINT UNSIGNED NOT NULL	否
sport	short	sport	SMALLINT UNSIGNED NOT NULL	否
dport	short	dport	SMALLINT UNSIGNED NOT NULL	否
time	double	time	DOUBLE NOT NULL	否
fileno	int	fileno	MEDIUMINT NOT NULL	否
offset	int	offset	MEDIUMINT NOT NULL	否
protocol	int	proto	MEDIUMINT NOT NULL	否

在程序中对 MySQL 数据库的操作是通过 libmysql-client 函数库提供的接口。使用 SQL 语句对数据库中的表进行插入和查询操作。通过性能测试，可以得出一个结论：关系型数据库不适合用来存储所设计的网包索引信息。

3.3.2　TM 系统

通用网络流量归档查询系统设计，主要考虑以下几个方面的设计：网包数据的采集，

网包数据的储存和索引。TM(Time Machine)系统[12][13]是一个典型的网络流量归档查询系统。

TM采用了网流截短的方案，取得了很好的节省存储空间效果。同时，TM又是开源项目，其系统设计和实现都有很多可以参考的地方。本节主要介绍为什么流截短的方案是可行的，以及TM系统的软件架构和实现细节。

1. 流截短方案的可行性

1）网流的分类

互联网上的流量可以被简单地分为两类，长流和短流。长流一般是大块数据的传送，如视频、音乐等多媒体的传输，文件的下载。短流一般是运行在网络上数量繁多的各种协议的信令交互、文本网页等。大的流量可以大至GB级别，小的流却可以不到1KB。记录网络流量的目的当然不是为了把互联网上传送的那些音频视屏、大型的文件给保存下来，而是要将在网络中运行的各种流量中有价值的信息记录下来。而这里，有价值的信息是指对事后分析、研究有帮助的信息，而数据传输显然不属于这一类。在众多的网包包含的信息中，按照对事后调查研究的帮助价值由强到弱，可以分为4种。

(1) 存在信息：一个流是否存在，IP地址、端口、协议，这部分是已有的成熟的网流信息归档查询分析系统所采集的信息。

(2) 交互信息：蕴藏在网包载荷中的信息，不同协议有着不同的交互命令，交互信息指的是能确定一个流的主要目的的信息，如是什么应用层的协议，要进行的是什么操作(比如HTTP GET、FTP STOR等)。

(3) 内容信息：蕴藏在网包载荷中的信息，指交互的具体内容(如HTTP GET请求的具体网页、服务器返回的错误信息、FTP STOR的文件名称、邮件内容等)。

(4) 数据信息：蕴藏在网包载荷中的信息，指纯粹的、大量的数据传输(比如网页上的视频和P2P文件下载等)。

在网络数据的传输中还有这样的特点：大块数据的传送总是在一些控制指令传送之后。换句话说，在一个较大的流中(MB级)，流的前部分更可能包含控制信息，而后面的部分很可能只是纯粹的数据。因此，当人们不得不舍弃一些网包时，采用流截短的方案能最大限度地保存下网流中的原始信息，被舍弃的主要是一些价值不大的数据信息。

2）网流大小的实际分布

实际网络流量的网流大小分布呈明显的长尾状，示意图如图3.10所示。图中横轴为流的编号，所有流按照大小顺序排序，大流排在前；纵轴为对应的流量大小。这种长尾分布意味着在对网络资源的占用方面，网络的大部分资源被一些较大的流占据；在网流数量方面，这些占网络大部分资源的流的数量与无数较小的流的数量比起来，又是很微小的。

图3.11取自TM[11]一文，图中的数据分别来自LBL(美国洛伦兹伯克利国家实验室)、MWN(慕尼黑科学研究网络)和NERSC(美国国家能源研究科学计算中心)。图中

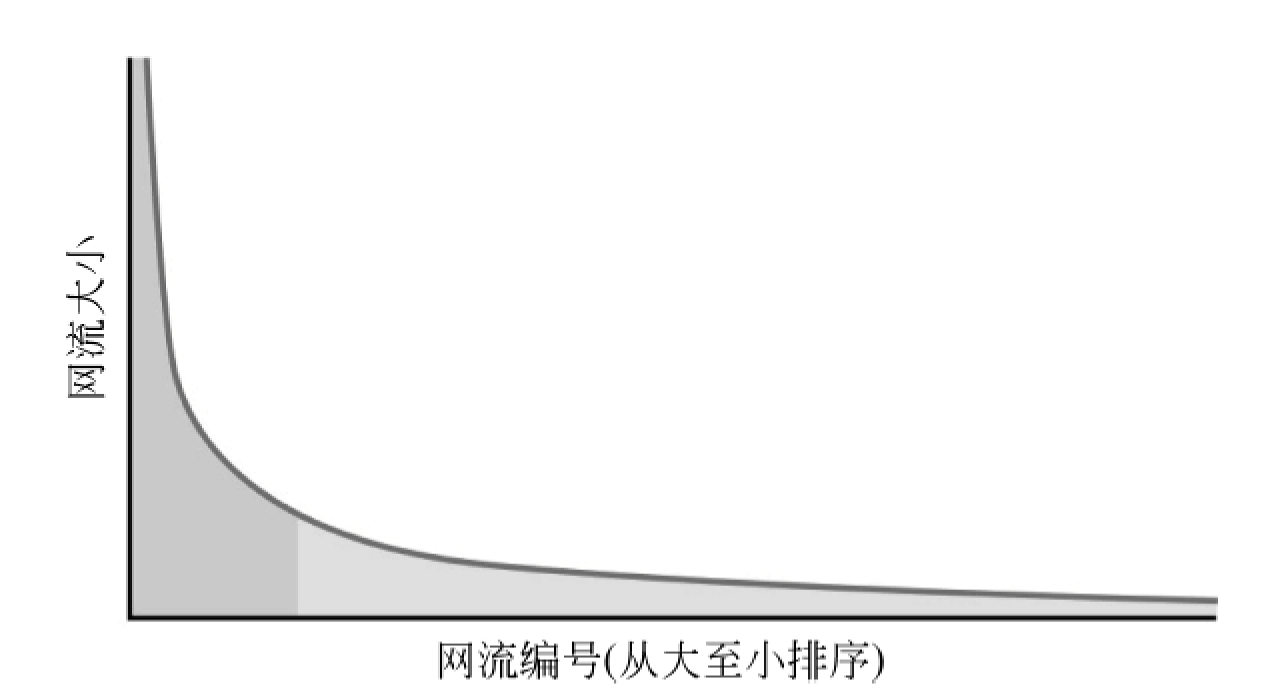


图 3.10　网络流量的长尾分布示意图

的横坐标是流的大小，纵坐标对应大于相应大小的流在数量上占总网流数的比重。可以看出，在 3 个数据集中，大于 10KB 的流所占比重均在 20%以下。其中，NERSC 的流量中含大流的文件较多，因为该网络内的服务器主要都用来做大型数据的存储，而基本没有什么网页应用，因此与 LBL 和 MWN 的数据特点差别较大。竖线位置对应着 20KB 的流，结果显示有 12%的 LBL，14%的 NERSC 和 15%的 MWN 的流量大于 20KB，而它们所包含的字节数却分别是数据集的 96%、99.86%和 87%。这预示着如果进行流截短，系统将可能处理相对于采取全记录策略时能记录的 10～100 倍的流量。

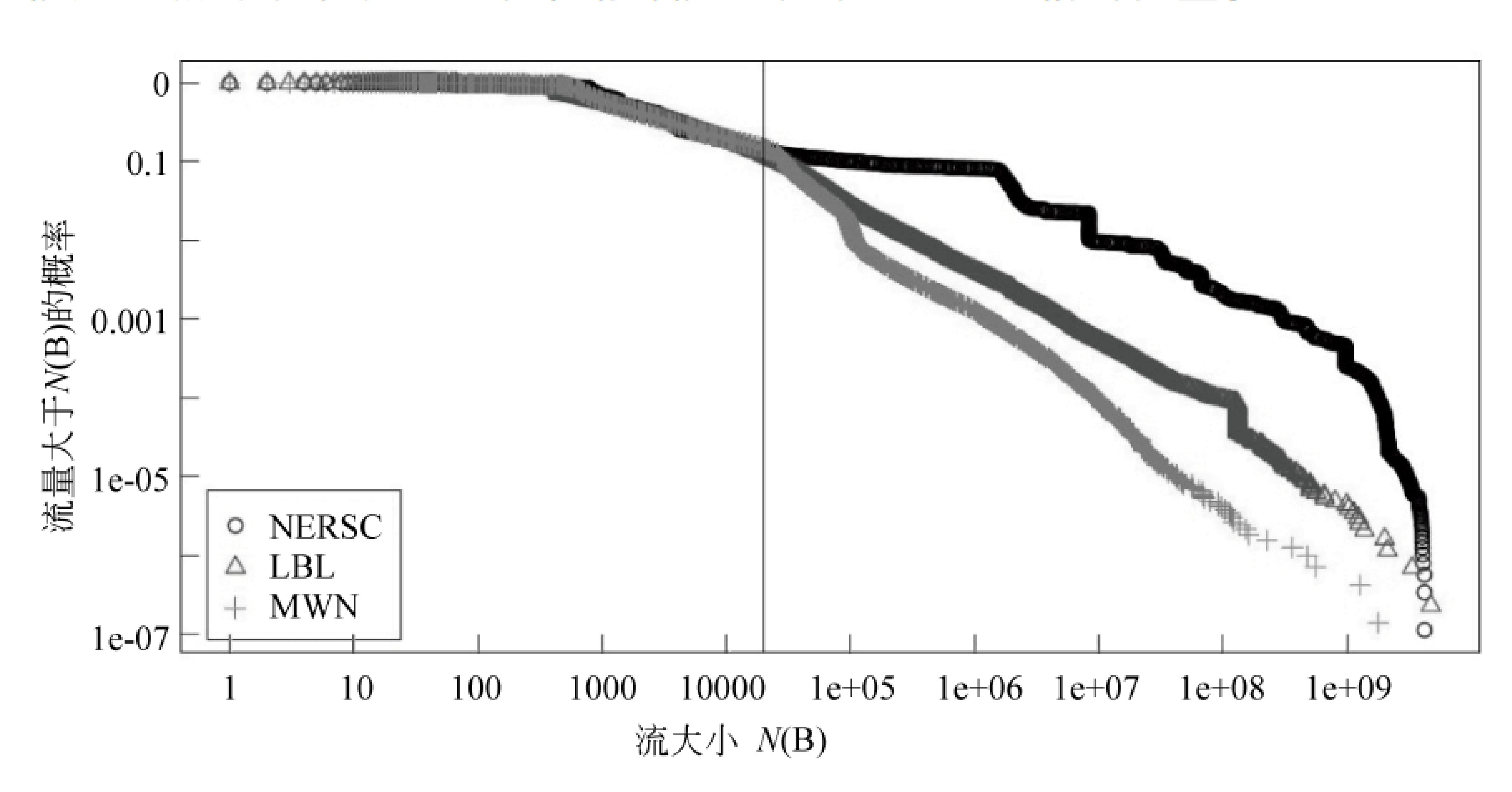


图 3.11　对数-对数流大小互补累积分布图

2. 软件架构

1）软件架构图

由于 TM 是开源项目，并且作者缺少开发类似系统的经验，于是首先对 TM 系统的实现进行了彻底的研究。TM 系统的架构如图 3.12 所示。

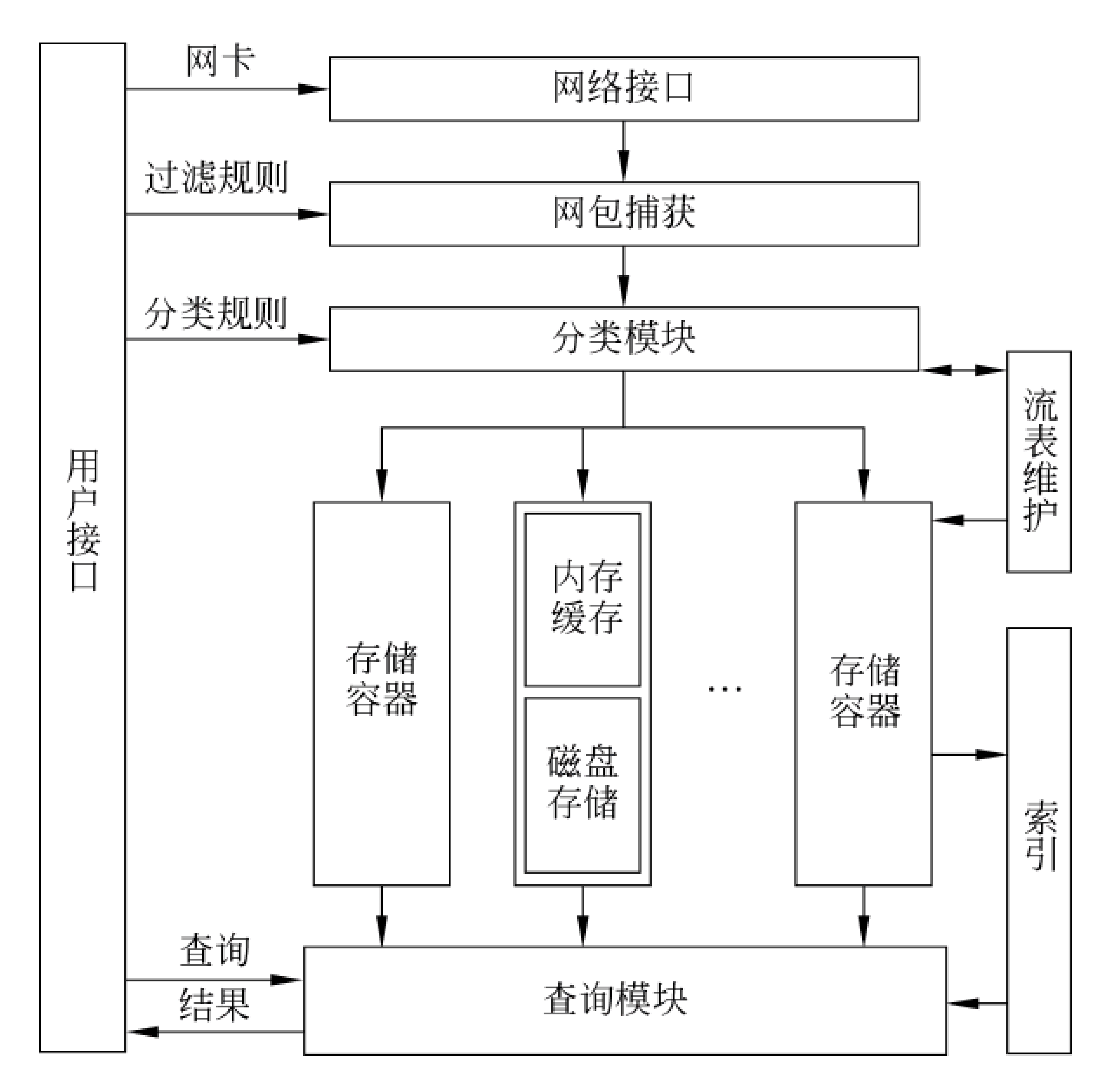


图 3.12 TM 系统的软件架构

从图 3.12 中可以看出系统主要分用户接口、网包捕获和分类、流表维护、存储容器和索引 5 个大块。用户可以通过用户接口实时地与系统进行交互,设定监听的网卡、过滤规则和分类规则,并进行网包的查询。系统从网络接口(网卡)捕获网包,网包经过分类后,被分配到相应类的存储容器。存储容器维护一个在内存中的缓存队列,当队列满时,把数据导出至磁盘。流表的两个主要功能是加速分类和实现流截短。流表中记录了一个流属于哪一个分类及目前通过的流量有多少等统计信息。存储模块将所存储的网包的信息交给索引模块,建立相关的索引,供查询模块使用。在下面会对每个模块会进行更详细的说明。

2)模块说明

(1)用户接口模块。

用户接口模块以命令行的方式和用户实现交互,首先读取用户的输入,并且做语法分析,将可以识别的命令进行执行。TM 中语法分析采用的是开源工具 yacc,该工具是一个 UNIX/Linux 环境下的编译器代码生成器(即用来生成编译器的编译器)。yacc 常与词法解析器 Lex 一起使用,生成语法解析器。该模块获取并解析用户输入后,调用相应的函数,将用户输入的命令和参数传给其他模块的接口执行,并将执行结果返回给用户。

(2)网包捕获和分类模块。

网包的捕获用的是 libpcap 库。libpcap 是 UNIX/Linux 上实现网络数据捕获的常用

库。大部分网络监控软件都应用这个库来实现网包的获取。关于 libpcap 的具体介绍在 3.1.3 节做了相关介绍。分类模块根据用户对类的定义(包含名称、特征和优先级等信息)将网包分至对应的类中。特征采用的是伯克利包过滤器(Berkeley Packet Filter, BPF)格式的字符串,可以对网包的基本信息进行描述,并且 libpcap 能直接在线编译执行该格式的过滤器。分类模块将网包递交给匹配过滤条件的优先级最高的类进行下一步处理。

(3) 流表维护模块。

流表记录的是网络中活跃的流的信息,对于每个流这些信息包括流的五元组信息、已经通过的字节数、分类模块分类的结果、流的起始时间、最近一个包的到达时间等。流表维护模块是为分类模块和存储容器模块提供支持。分类模块处理一个网包时,先在流表中查询是否有相应记录,如果有,则直接利用原有的分类结果,将网包递交给相应类的容器中;若没有,则在流表中新增一条记录并存入分类结果。存储模块收到一个网包时,首先在流表中查询其所属流的字节数是否已经超过截断点,若超过则仅对包的信息进行相应的统计操作,而不再将网包放入内存的缓存队列中。为了控制流表在内存中的大小,设置一个超时时间,每隔一段时间将那些超过超时时间仍未来新的网包的流量的记录从流表中移除。

(4) 存储容器模块。

由于内存和硬盘的读取速度相差几个数量级,频繁的磁盘写操作性能较低。因此存储容器模块在内存中维护一个缓存队列。当缓存队列满时,再将队列中的一块数据存入硬盘,腾出空间给新的数据。这样的批量操作能提高系统效率。

(5) 索引模块。

存储容器模块需要索引模块对存储的数据进行索引。为了以多线程的方式运行,索引模块维护一个待处理的队列。存储模块如果有要索引的网包就递交到这个队列中。索引模块分为两个部分,内存中保存的索引和写入到文件的索引。虽然人们很希望所有的索引信息都能保存在内存中,但内存大小的限制显然不允许这样的实现。查询时,查询模块利用索引模块返回的查询结果,到存储容器模块的内存队列和相应的网流文件中提取相关的网包,整合成一个文件返回给用户。

3) 线程说明

TM 软件中这些模块的运行是以多线程的方式并行的,共有 8 个线程。其中用户界面是一个线程,统计和日志是一个线程,网包的抓取和分类是一个线程,4 种不同类型的索引每种索引类型分别是一个进程,索引的聚合是一个线程。

3. 软件实现细节

本节将介绍几个重要模块的实现细节。索引模块是研究的重点,因此下面单独介绍。

本节介绍流表维护模块和存储容器模块中内存中缓存的管理以及数据导出到硬盘的机制。

1）流表的维护

流表其实是一个散列表，对于一个流，将其的五元组信息进行散列，得到一个整数，再将这个整数对散列表的表长取模，就得到一个流的记录应该在散列表的位置。如果两个流的五元组信息不一致，但却又分在了散列表的同一个位置，则出现了碰撞。因此散列表内的每个位置指向的都是一个碰撞队列，队列中的元素散列值可能一样也可能不一样，但是对散列表长取模后都被分在了相同的位置。

查询某个流是否存在时，首先获取这个流应该在散列表的什么位置，然后判断这个位置上是否有流记录。如果有则在该位置的碰撞队列顺序查找，逐个比对五元组信息是否完全一致。如果未找到五元组信息完全一致的流记录，则表示这是一个新出现的流，在流表中并不存在。

如果流表中不存在相应的信息，则需要在流表中插入新的流记录。这时将流记录添加到该流五元组对应的散列表的相应位置所指向的碰撞队列中。

流表向网包分类模块返回流信息结构的指针，后者利用该指针对流信息做更新。

流表会定时删除其中超时的流记录。超时时间为用户制定，记为 timeout，若流信息中显示最近一个网包在 t 时来到，而此刻时间为 T，则当 $\text{timeout}+t<T$ 事该流被视为超时。

2）内存中的缓存队列

TM 的存储容器模块管理的缓存队列是一个 FIFO 队列(First In First Out)。队列实际上是一块很大的连续内存空间。要存储的网包一个接一个地连续地存放在这样的一块内存空间中，当内存空间写尽后，将内存空间的一部分导入硬盘，释放出新的可用的空间。具体实现如图 3.13 所示。

程序中记录内存的起始地址 start，开辟一大块内存空间后，在最后一段预留出一个最大包所占空间，并记录位置 end，如图 3.13(a)所示。同时维护指针变量 sp、lp 和 wp。其中，sp 表示队列中最旧的有效网包所在位置，lp 表示上一个包写入的位置，wp 表示当前包写入的位置。

起始时，队列中没有数据，sp、lp、wp 均指向 start 位置。随后，当网包到来时，每写入一个网包，lp 被赋值为 wp，而 wp 指针向后移动刚写入的包长，如图 3.13(b)所示。

当写入网包后，wp 超过 end 位置时，wp 回卷到 start 位置，如图 3.13(c)所示。

当 wp 回卷到 start 位置后，长度为 l 的新的网包到来时，检查 wp 和 sp 的位置，如果 $\text{wp}+l>\text{sp}$，意味着如果继续写入原有的有效数据将会被覆盖，因此需要导出一部分队列中的数据，给新的包腾出空间。所以从 sp 开始，导出 sp 后面的 n 个包，并将 sp 指针后移到未导出的最旧数据的起始位置，如图 3.13(c)所示。之后继续写入新的包，知道下一次

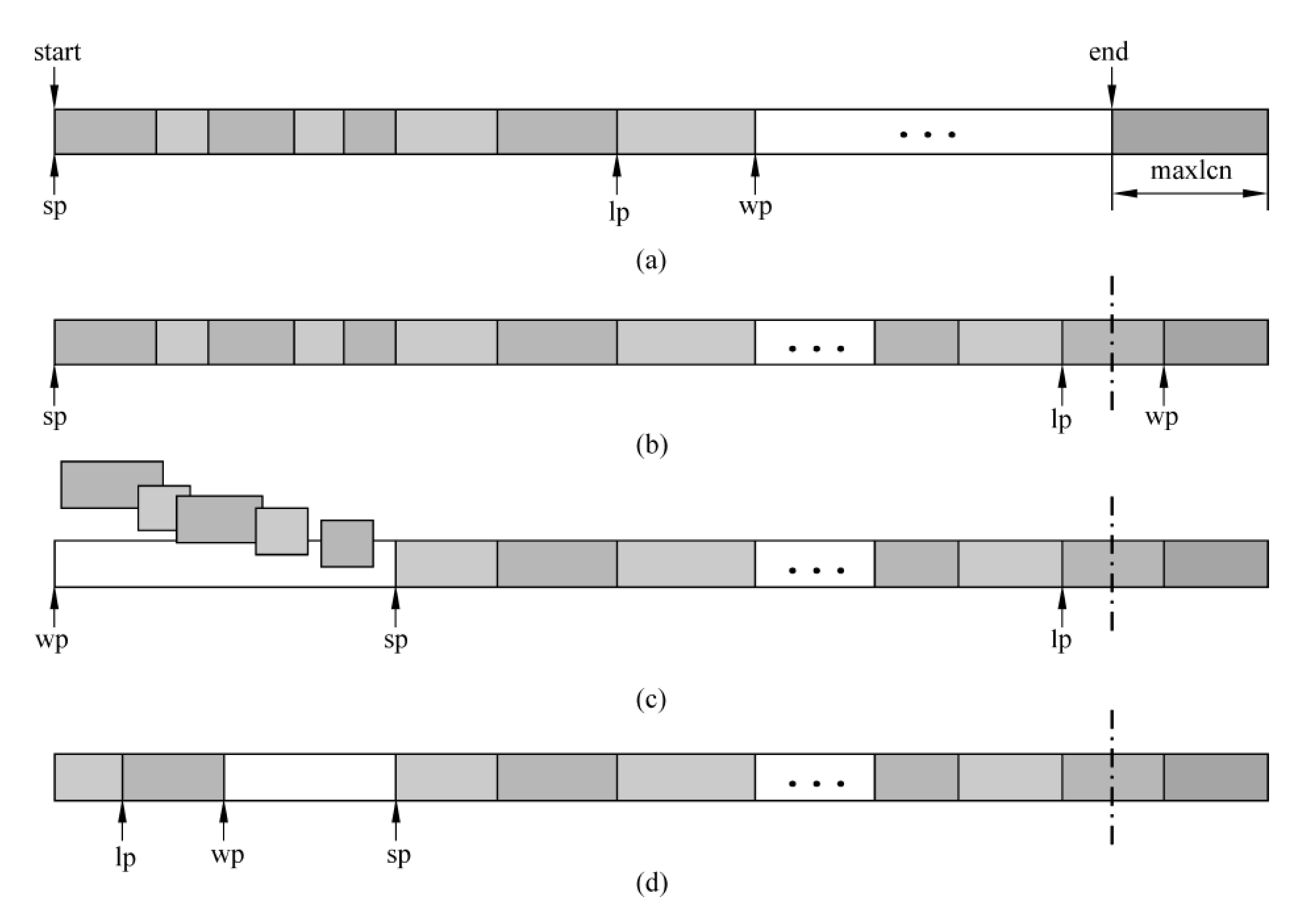


图 3.13　TM 系统内存中的 FIFO 队列维护示意图

wp+l>sp 发生时，再进行队列中旧数据的导出，如图 3.13(d)所示。

在内存中自己维护一个连续内存的 FIFO 队列的好处是，不用频繁进行内存的分配与释放，相比用链表维护 FIFO 队列，效率更加高。

3）将记录转移到硬盘

当内存中的 FIFO 队列满时，要向磁盘导出一批旧的数据，这需要对硬盘文件的读写操作。由于网络流量很大，因此势必要对文件大小做限制，不然将带来查询时文件读写的巨大负担。

TM 允许用户设置单个文件的大小上限，当进行写操作的一个文件大小超过这个上限时，就会关闭该文件，另开一个文件继续写操作。文件名为“class_类名_8 位的十六进制数”。对每一个类，该数字从 1 开始，逐次增加。存储效果如图 3.14 所示。

4. 网流索引

前文介绍 TM 一共有 5 个线程与索引有关，也介绍了 TM 是如何在内存中缓存包以及当内存中的缓存队列满时，将流量存入到规定大小的文件中。本节详细介绍 TM 是如何为内存缓存队列以及磁盘上的文件做索引并根据该索引进行查询。

1）索引概述

TM 中建索引是根据流活跃的时间，记录流活跃的期间。活跃的区间是一组区间的

```
root@saturn-desktop:/media/disk/rly/shouxin_data/tm_backup/2011-03-31-0# ls -lah
total 101G
drwxr-xr-x  4 root root  20K 2011-03-30 23:42 .
drwxr-xr-x 56 root root 4.0K 2011-05-22 10:52 ..
-rw-r--r--  1 root root 256M 2011-03-28 21:44 class_all_00000c6f
-rw-r--r--  1 root root 256M 2011-03-28 21:59 class_all_00000c70
-rw-r--r--  1 root root 256M 2011-03-28 22:14 class_all_00000c71
-rw-r--r--  1 root root 256M 2011-03-28 22:29 class_all_00000c72
-rw-r--r--  1 root root 256M 2011-03-28 22:47 class_all_00000c73
-rw-r--r--  1 root root 256M 2011-03-28 23:04 class_all_00000c74
-rw-r--r--  1 root root 256M 2011-03-28 23:21 class_all_00000c75
-rw-r--r--  1 root root 256M 2011-03-28 23:39 class_all_00000c76
-rw-r--r--  1 root root 256M 2011-03-28 23:59 class_all_00000c77
-rw-r--r--  1 root root 256M 2011-03-29 00:18 class_all_00000c78
-rw-r--r--  1 root root 256M 2011-03-29 00:39 class_all_00000c79
-rw-r--r--  1 root root 256M 2011-03-29 01:03 class_all_00000c7a
-rw-r--r--  1 root root 256M 2011-03-29 01:25 class_all_00000c7b
-rw-r--r--  1 root root 256M 2011-03-29 01:45 class_all_00000c7c
-rw-r--r--  1 root root 256M 2011-03-29 02:07 class_all_00000c7d
-rw-r--r--  1 root root 256M 2011-03-29 02:32 class_all_00000c7e
-rw-r--r--  1 root root 256M 2011-03-29 02:57 class_all_00000c7f
```

图 3.14　TM 存储的网流文件

集合，比如某个 IP 的活跃区间是(a，b)、(c，d)、(e，f)。查询时，得到流活跃的期间，再利用时间区间的信息在内存和文件中查找对应的网包。TM 中维持 4 种不同的索引，它们都继承自索引类，每一种索引都有单独的进程。同时全局中还有一个索引聚合线程每隔一段时间触发一次每种索引的聚合操作。4 种支持的索引如图 3.15 所示，分别包含不同含量的信息：单元组索引只包含一个 IP 地址，也就是说，无论源地址或是目标地址为 a 的网包，即使它们属于不同的网流，如果采用单元组索引，都会被记录到 IP 地址 1 为 a 的索引项。分成 4 种索引的好处是查询时不用再做集合的归并操作，对于各式各样的查询，能够通过索引中存储的记录，直接得到结果；但是同时也带来不好的影响，即增加了计算负担。

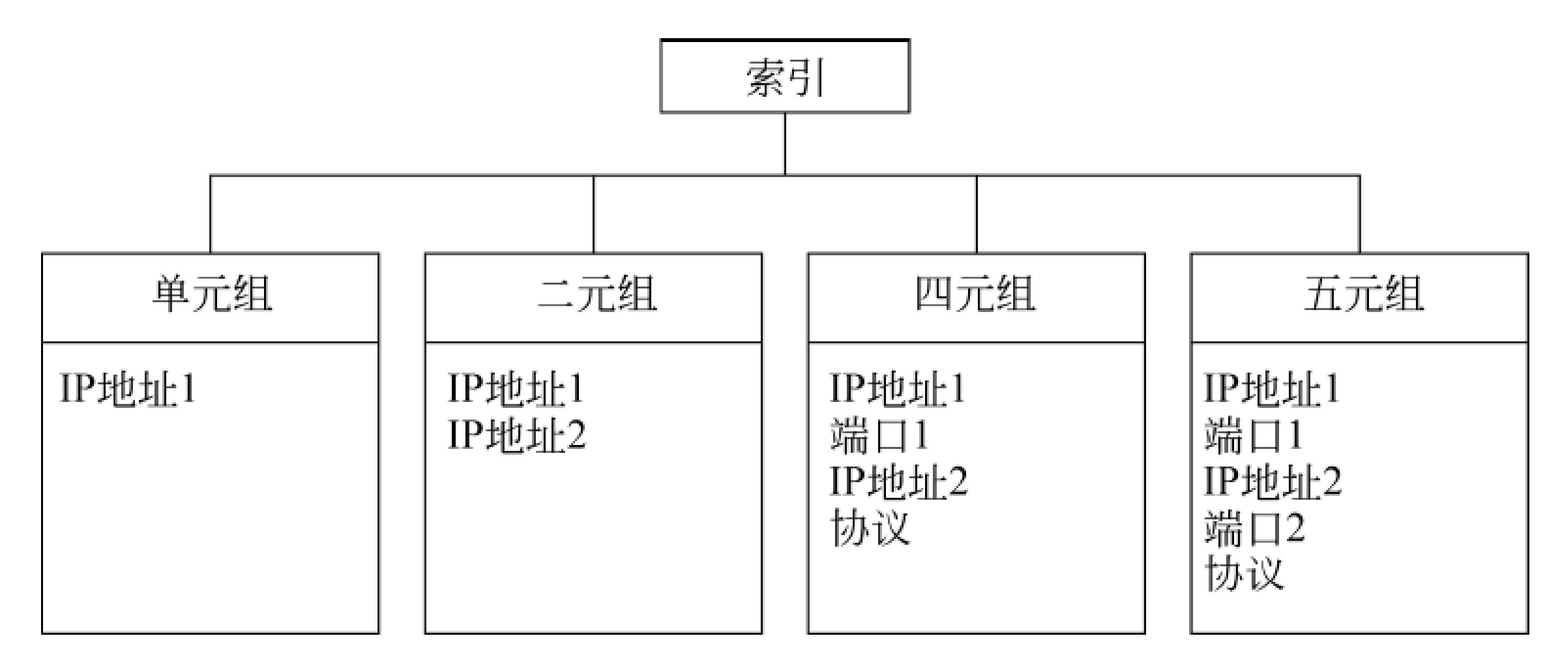


图 3.15　TM 中的索引类型

由于索引线程和包处理线程是相互独立的两个线程。因此每个索引线程都维护一个队列。当包处理线程处理一个网包时，会将网包投送到每一个索引线程的队列中。索引线程是一个读取队列中的网包并进行索引更新的循环，直到程序结束才停止。

像网包一样，由于内存容量的限制，索引不可能都存放在内存中，必须定时地向硬盘导出。本节将分别介绍内存中的索引管理和导出机制。

2）缓存在内存中的索引

对应一种索引类型 T，维护索引 T 的线程 T_thread 维护一张散列表，散列表中存放着已经建立的记录。每一条记录中含有该记录对应的索引变量 t，一组 t 所对应流的活跃时间区间的集合，以及一些统计信息。

每次 T_thread 从队列中获取一个网包，并且将该网包的信息进行提取，获取对应的索引变量 t 后，首先在散列表中查找索引变量为 t 的记录是否已经存在，若不存在则在散列表中新建一条索引为 t 的记录。之后利用网包到达的时间 time 对相应的记录进行更新。设该记录中离 time 最近的时间跨度为(a,b)，则更新规则如下：首先判断 time-b 是否小于阈值 f，f 是事先设定的常数，若小于则将最新的时间跨度更改为(a，time)，否则新加一个时间跨度(time，time)。

3）索引向硬盘的导出

虽然像网包一样，索引也不能在内存中长久地保存，但是索引的导出条件却与网包的导出发生的条件截然不同。网包的导出发生在内存中用来缓存网包的 FIFO 队列没有空间时，而 TM 并没有对索引在内存中的大小加以限制。索引的导出发生在存在与内存 FIFO 队列的最旧记录的时间比上次索引导出的时间还要新的时候。这时候意味着现在内存中的索引对应的流量很多已经不在内存中了。

导出后将清空散列表，同时根据反馈信息调整新的散列表的大小。如果原散列表的大小为 size，散列表中存储的记录的个数为 num，则有：如果 size＞2×num，则新的散列表大小缩减为 size/2；如果 num＞2×size，则新散列表的大小扩大为 2×size；其余情况散列表的大小不变。

导出后旧的散列表中包含的信息将写入磁盘文件中，格式如下：文件开头 16B 是两个 double 类型的数，分别代表该索引的起始时间和结束时间，之后的 4B 是一个 int 类型的数，表示该索引文件中使用的索引类型的数据结构占的字节数。之后的全部文件连续写入的是(索引类型变量、起始时间、终止时间)这样的结构体，一直到文件的结尾。需要注意的是散列表中一个索引下的活跃时间区间集合中的区间如果大于一个，所有的时间区间都将按上面所说的格式连续地写入索引文件中。导出文件的文件名类似流文件的命名方法，为“索引类型_聚合等级_8 位十六进制编号”。

4）索引的聚合

为了避免索引文件序号的飞速增长，TM 设计了索引的聚合机制。全局中有一个索引聚合的线程，定时调用各个索引线程的聚合方法。聚合的目的就是将几个小的索引文件合并为一个大的索引文件。在索引文件名中有一个区域代表聚合等级。10 个第 n 级的文件会聚合成一个 $n+1$ 级的文件。实际结果如图 3.16 所示。

```
root@shouxin#
/data/tm_backup/2011-03-31-0/indexes # ls
index_connection2_00_00001c20  index_connection4_00_00001c20
index_connection2_00_00001c21  index_connection4_00_00001c21
index_connection2_00_00001c22  index_connection4_00_00001c22
index_connection2_00_00001c23  index_connection4_00_00001c23
index_connection2_00_00001c24  index_connection4_00_00001c24
index_connection2_02_00000040  index_connection4_02_00000040
index_connection2_02_00000041  index_connection4_02_00000041
index_connection2_02_00000042  index_connection4_02_00000042
index_connection2_02_00000043  index_connection4_02_00000043
index_connection2_02_00000044  index_connection4_02_00000044
index_connection2_02_00000045  index_connection4_02_00000045
index_connection2_02_00000046  index_connection4_02_00000046
index_connection2_02_00000047  index_connection4_02_00000047
index_connection3_00_00001c20  index_ip_00_00001c20
index_connection3_00_00001c21  index_ip_00_00001c21
index_connection3_00_00001c22  index_ip_00_00001c22
index_connection3_00_00001c23  index_ip_00_00001c23
index_connection3_00_00001c24  index_ip_00_00001c24
index_connection3_02_00000040  index_ip_02_00000040
index_connection3_02_00000041  index_ip_02_00000041
index_connection3_02_00000042  index_ip_02_00000042
index_connection3_02_00000043  index_ip_02_00000043
index_connection3_02_00000044  index_ip_02_00000044
index_connection3_02_00000045  index_ip_02_00000045
index_connection3_02_00000046  index_ip_02_00000046
index_connection3_02_00000047  index_ip_02_00000047
```

图 3.16　TM 索引的聚合结果示例

5. 查找过程

当用户输入查询请求时，用户接口线程对用户的输入进行分析，判断用户是利用哪一种索引进行查询。查找的过程分为两个步骤：第一个步骤是从建立的索引中提取对应网流的活跃时间区间，完成这个步骤要对内存中的索引和磁盘上的索引进行搜索；第二个步骤是利用得到的时间区间，到相应的内存缓存队列和磁盘文件中提取要查询的流量。假设用户使用索引类型 T，要搜索的网包的索引变量为 t，下面简述每一过程的实现。

1）从内存中的索引提取时间区间

在管理 T 索引线程所维护的散列表中查找 t，若不存在则返回；若存在则记录下 t 所对应记录的活跃时间区间集合中与要查询的时间区间有重合部分的区间。

2）从硬盘上的索引文件提取时间区间

在所有的 T 索引导出的索引文件中，首先根据时间判断是否有必要遍历文件，如果该文件的时间区间与想要查找的时间区间没有交集，则跳过该文件；如果不能判断是否存在交集，则需要对该文件进行遍历，找到 t 所对应记录所记载的时间区间，记录下那些与要查询的时间区间有重合的时间区间。

3）从内存缓存队列中提取匹配的网包

将上两步的结果合并后，就可以得到一个总的时间区间集合，要查找的对象一定存在于这些时间集合中。利用这些集合可以缩小需要比对的网包记录的集合。

首先在内存缓存队列中提取匹配的网包。对应一个要查询的时间区间(s,e)和一个时间集合中的时间区间(a,b)，因为两个区间有交集，显然满足 $s<e, a<b, a<e, s<b$。此时首先判断(a,b)是否与内存队列中所含网包的时间区段(t_oldest，t_newest)重合，t_

oldest 是内存缓存队列中最早的网包，t_newest 是内存队列中最新的网包，若没有交集则忽略，若有交集则说明内存队列中可能包含该网包，需要对内存队列进行搜索，步骤如下。

首先判断搜索的起始点，如果 $a \leqslant$ t_oldest，则从缓存队列中最早的有效包开始逐个搜索，逐一检查，如果匹配直到某个包的时间超出查询范围或查完队列中最后一个包。如果 $a >$ t_oldest，则要首先在内存中确定时间戳在 a 之后的第一个包。由于从队列第一个有效包开始顺序查找并不是最优的方案，TM 设计了很巧妙的“近似折半搜索”来加速搜索速度。注意到内存中的缓存队列是一块连续的数据，仅有的指针分别指向下次写的位置、上次写的位置和最早的有效包的位置，而内存中的数据并没有统一的大小，有的包长，有的包短。如果之前没有记录包的起始位置，只有从最早有效包的位置开始向后遍历，才可能获得其他包的位置。但是如果事先将所有包的位置记录下来，开销比较大，意味着还要维护一个指针的 FIFO 队列。TM 设计了一种折中的方案，将内存分为很多格，每格有特定的长度，每格关联一个指针，记录这一格中第一个网包所在的位置。“近似对半搜索”时，利用这些格关联的指针，确定离 a 最接近的但比 a 早的那一个指针所指的包，从这个指针所指的位置开始向后搜索。算法示意图如图 3.17 所示。

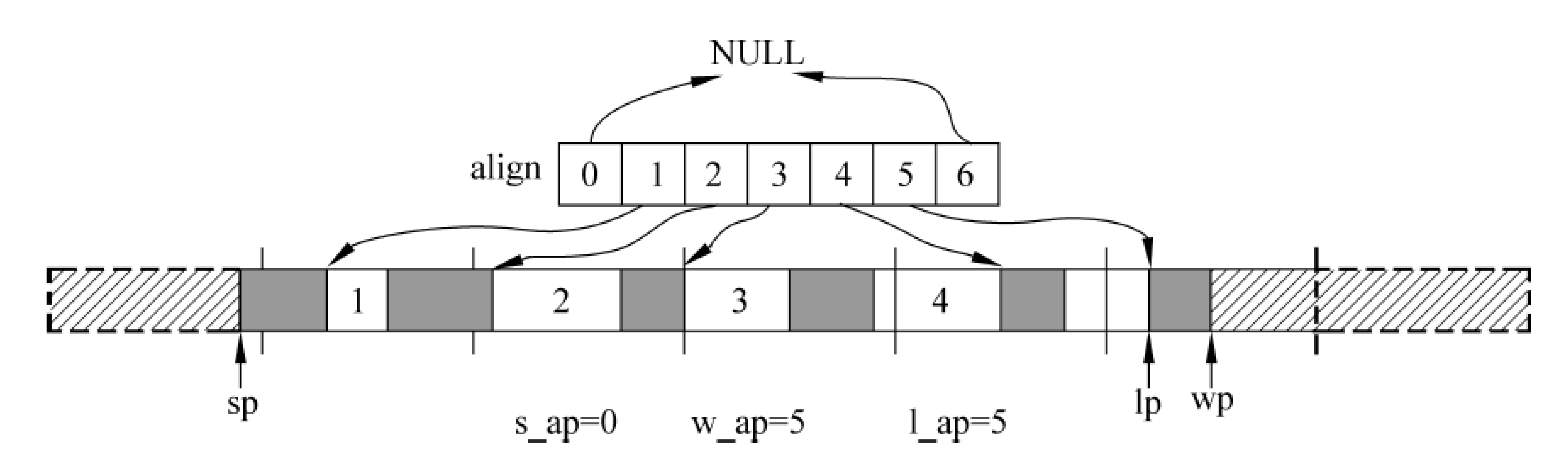


图 3.17　TM 近似对半搜索算法示意图

图 3.17 中，s_ap、l_ap、w_ap 分别表示 sp、lp、wp 所在的格，浅绿和蓝色表示在内存中顺序存放的网包。align 指针数组中第 n 项指向第 n 格的第一个包。如果第 i 格的开始包含于无效区域中，则 align 数组第 i 个元素指向 NULL。无效区域是指已经释放的为新的写入预留的空间，里面的信息均会被覆盖，对应于红色斜纹所示部分。近似对半搜索就是指的是在 align 数组中的指针所指的元素中进行对半查找，找到其中时间戳离 a 最接近且比 a 早的一项。

4）从磁盘文件中提取匹配的网包

遍历磁盘上的网包导出文件，判断要查询的时间区间是否与一个网包文件包含的时间跨度重合，如果不重合则查下一个文件，直到所有文件遍历完，如果有重合的部分则跳到重合部分的开始，依次遍历剩下文件中的网包，逐一判断是否匹配搜索条件，如果匹配则提交给搜索结果，直到所查的网包时间超出要查询的区间或搜索到文件尾。

从内存缓存中和从磁盘文件中搜索出的匹配查询要求的网包将合并在一起，以查询

语句中指定的名字输出到查询结果目录中。

6. 系统性能

1）存储性能

本章中涉及的实验数据集来自北京市首信公司某 IDC 机房的出口网关镜像流量，大小为 102GB，在后文中，我们简称为 TraceA。TraceA 由许多个 100MB 级的文件组成。

为了测试 TM 的存储性能，作者改写了 TM 的网包读入模块，使其能够连续地读入网包文件，以 TraceA 作为输入进行测试。在离线模式下 TM 会尽最大速度处理网包，其处理 TraceA 所花时间如表 3.4 所示。

表 3.4　TM 处理性能测试结果

测 试 样 本	TraceA
流量文件大小	102GB
网包数	422 505 000
处理时间	4971s
处理速率(按字节)	20.5MB/s
处理速率(按网包个数)	84 994 Packets/s

2）查询性能

针对 TraceA，我们分别构造两组查询，分别表示为 Q1、Q2，每组查询集中均有 10 条不同的查询语句。其中一组中所含查询都是全局查询，即不给出时间范围，二组中的查询均给出小于 600s 的时间区域。测试结果如表 3.5 所示。

表 3.5　TM 查询性能测试结果

单位：秒	TraceA 查询耗时	
查询编号	Q1	Q2
1	5.42	1.85
2	1.82	0.26
3	3.12	2.87
4	5.06	0.24
5	14.92	0.16
6	2.72	0.22
7	2.23	0.26
8	7.14	0.21
9	82.15	0.07
10	2.74	0.15

3.3.3　TIFA 系统实现[14]

本文主要研究 100Mbps～1Gbps 级的网络环境中，流量归档查询系统的设计与实现。软件运行在 x86 平台 Linux 系统上，能在普通配置的计算机上实现对目标量级的网流的记录、索引和在线查询的功能。

1. 网络流量数据特点

网包的索引信息具有以下一些特点：海量、数据结构固定、只增不改、重复性较高。海量是指网包索引信息条数众多，一天可以产生几百万条甚至上亿条索引信息。数据结构固定是指每一条网包的索引信息都有固定的格式和固定的长短。只增不改是指网包的索引信息只会不断增加，一旦产生，以后不可能也不需要再进行修改。重复性高指就每一个域来看，一个域中的千万条数据出现大量的重复。这些特点导致使用关系型数据库处理这样的数据效率并不高。

2. 基于 FastBit 的 TIFA 系统实现

本节提出的 TIFA(TImemachine＋FAstbit)系统设计，利用 FastBit 的位图索引数据库，进行网包索引信息的建立、存储与查询。FastBit[13]是一个开源的支持 SQL 语言的位图索引数据库的实现。

FastBit 并不提供网流处理的程序接口，编译 FastBit 源代码后得到可执行文件，需要通过命令行参数的形式和位图索引数据库进行操作。FastBit 将一个文件夹作为一个表，文件夹中有该表的定义说明。在一个表中，每一列都是单独的一个文件，除此之外每一列还有一些 FastBit 生成的索引文件。

存储时，指定包含数据的 CSV 格式的文件以及存储时各个域的定义和存储的表的位置。CSV 格式是使用逗号将各列元素分开的格式。FastBit 接收命令后，利用各个域的定义从 CSV 文件中读取相应的数据，并将数据增加到指定的表中。查询时，指定相应的表和查询的 SQL 语句以及输出的方式，FastBit 将从指定的表中搜索符合查询条件的记录，并以指定的输出方式将结果以 CSV 文件的形式保存在指定的文件中或打印在屏幕上。

由于一个 CSV 文件中可以存放多条记录，因此可以实现一次性导入多条数据，这样可以减小以命令行形式对 FastBit 数据库进行操作带来的多余时间消耗。系统流程如图 3.18 所示。产生一条索引后，首先将索引存储到索引缓存中，累积一定的条数后，将缓存中的索引记录以 CSV 格式写入一个文件，并利用 FastBit 提供的命令将文件中的内容添加到 FastBit 数据库的表中。处理完后清空缓存，循环使用。同时维护 FastBit 数据库中的表，使得每一个表中的记录条数不至太大。当一个表中的条数达到一定界限时，重新建立一个表。系统维护每一个表所拥有记录的时间范围，这样在查询中，可以通过时间区

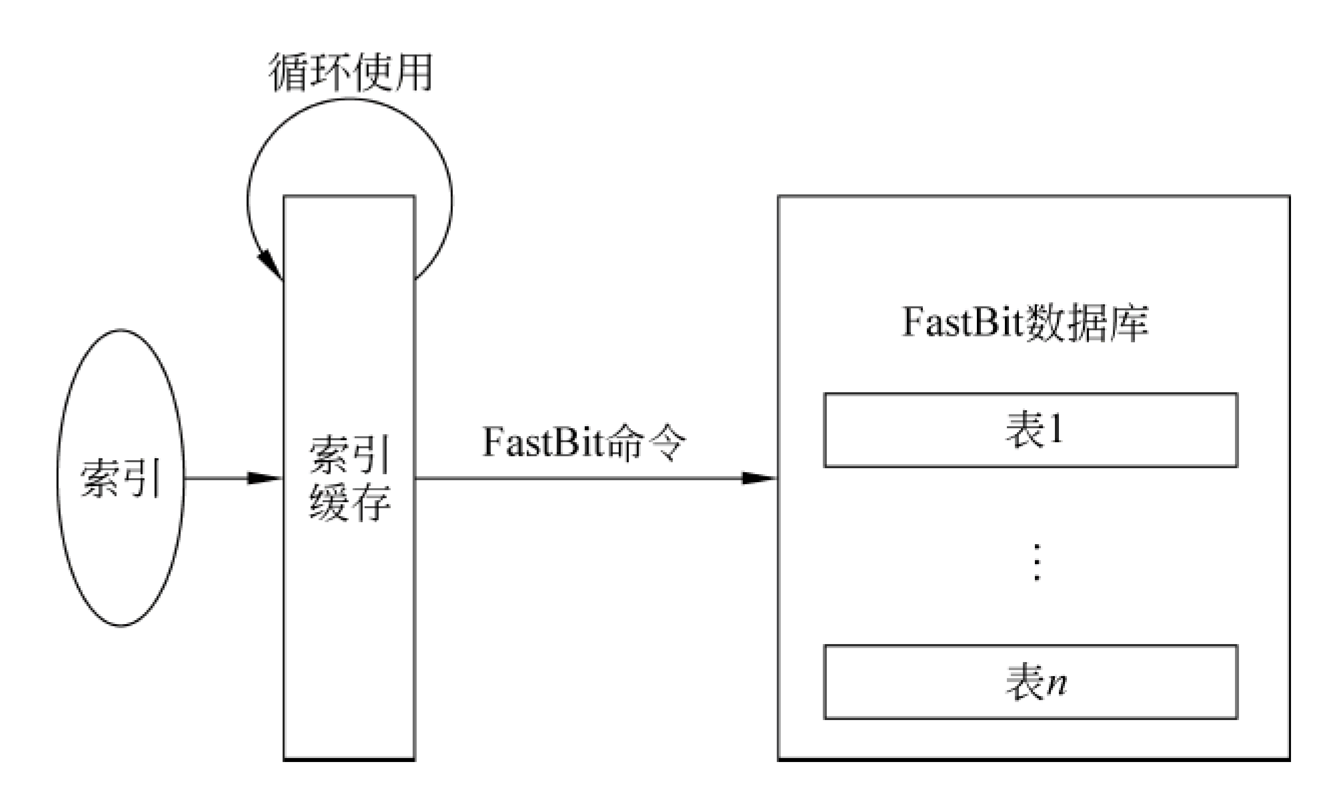


图 3.18 使用 FastBit 数据库进行索引信息存储的流程

间过滤掉一些不必的搜索。

查询时，利用查询的时间段确定需要进行搜索的表，并对这些表利用 FastBit 进行查找，将查询结果输出到指定文件，再把文件中的内容读入内存，解析出文件编号和在文件中的偏移量等信息，从相应位置提取网包。

3. 基于 FastBit 的 TIFA 系统性能

同样对使用 FastBit 数据库所实现的系统（下文简称为 FastBit 方案）进行处理性能和查询性能的测试，使用 TraceA 和 Q1、Q2 两个查询集，得到结果如图 3.19 和图 3.20 所示。

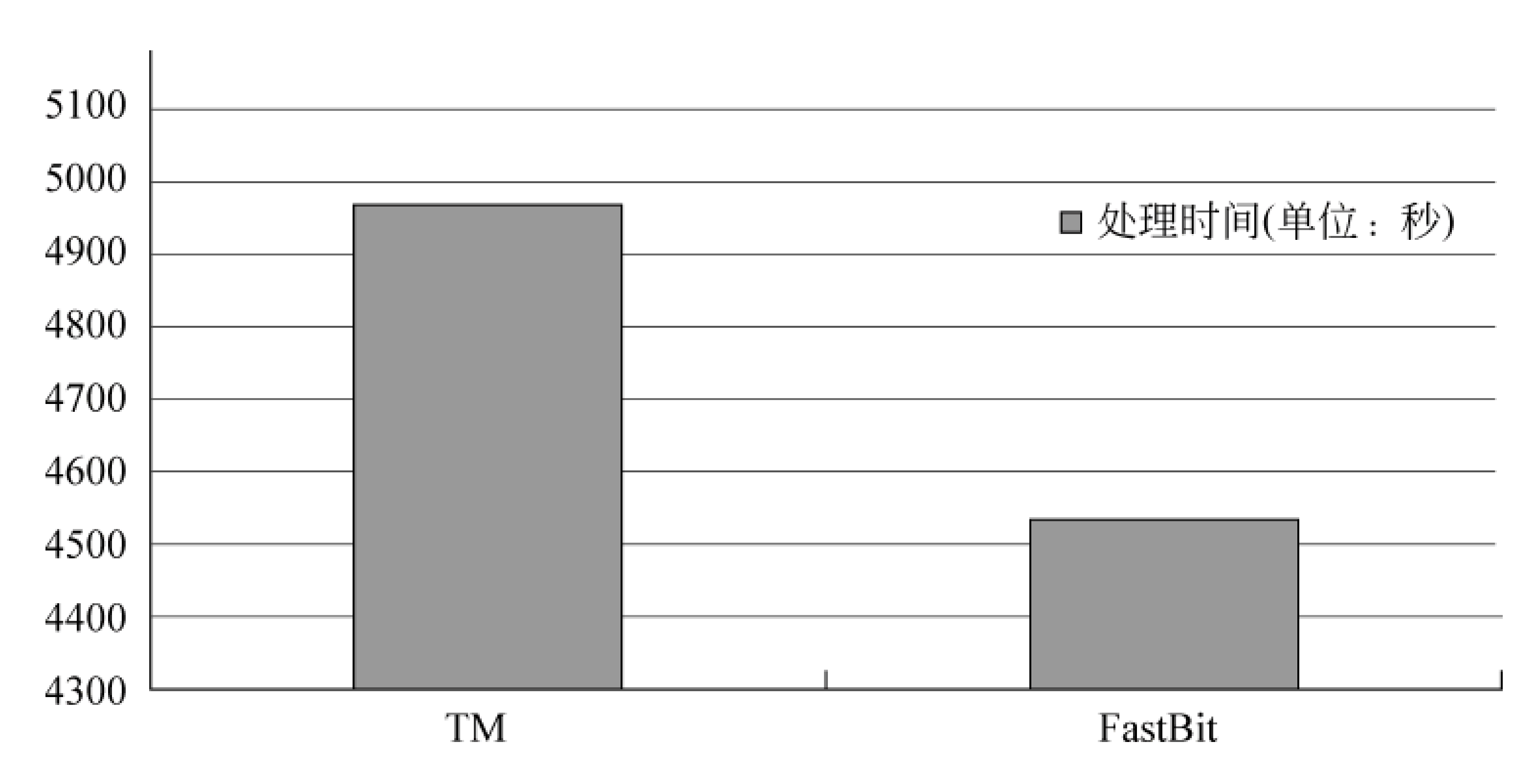


图 3.19 FastBit 方案与 TM 存储性能对比

在存储性能方面，FastBit 方案比 TM 约快 10%，同时 TM 由于超出了处理速度造成了一些索引的丢弃，而 FastBit 对每个包都建立了一条索引。使用 FastBit 作为存储网包索引的数据库并且采用批量存储的方案在存储性能方面表现出较高效率。

在查询性能方面，FastBit 也展现了较高的效率。对于查询集 Q1 和 Q2，FastBit 方案

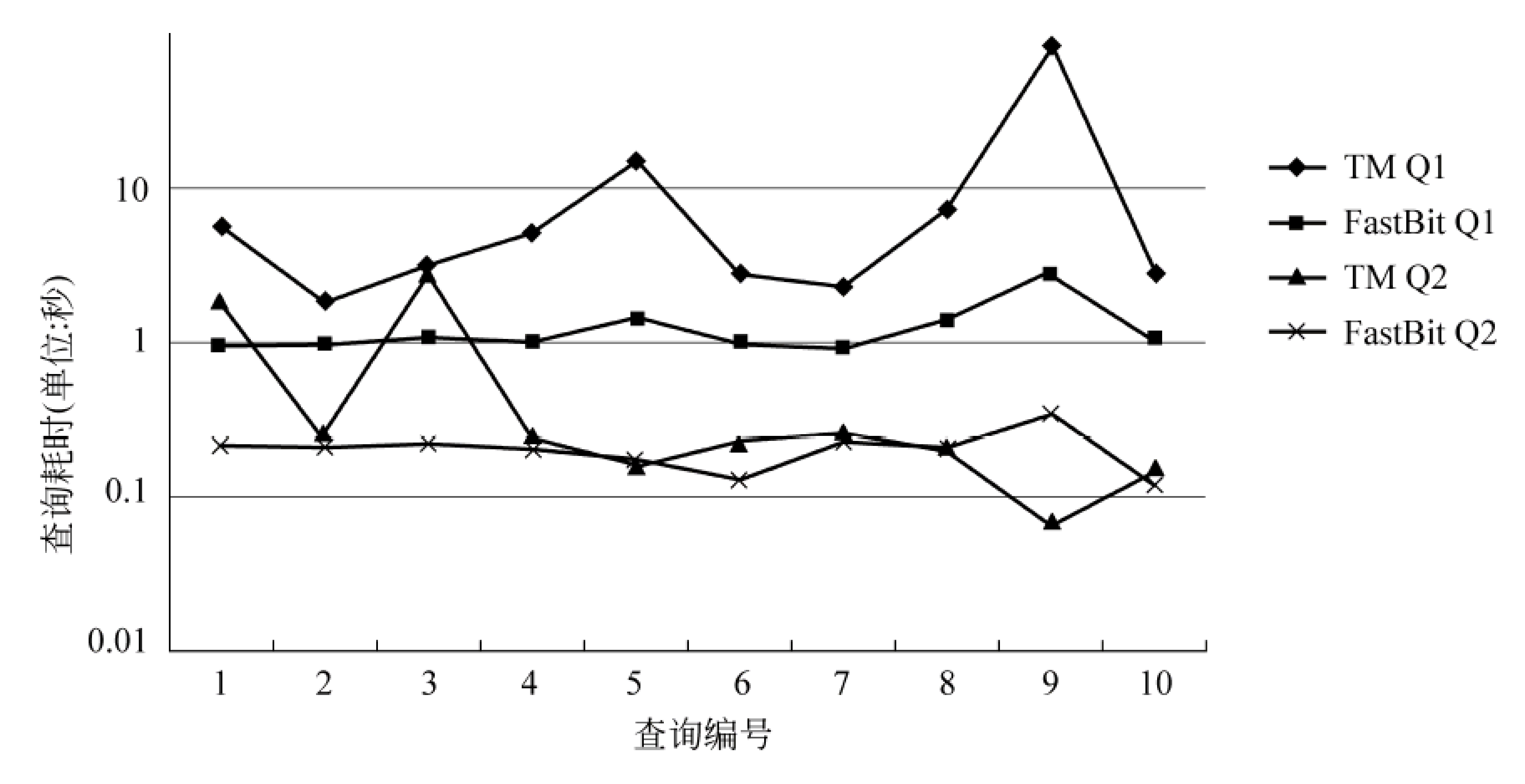


图 3.20　FastBit 方案与 TM 查询性能对比

的查询速度均比 TM 要快。需要注意的是在 Q1 和 Q2 查询集的第 9 个查询点上，表现了一些与其他点相异的特征。Q1 查询集得第 9 个点相比其他点所花时间高出许多，这是因为这个点匹配的查询结果比较大，最后生成的查询结果约 23MB，因此花费时间较多。在 Q2 查询集中，第 9 个点在处理上由于输入数据的失误，在流文件中并不存在匹配的记录，这是 TM 比 FastBit 方案快的原因，是因为 TM 的索引记录的是一个流的活跃信息。如果 TM 在查询条件中指定的时间段里不存在这个流的记录，就不会再进行进一步的搜索。

通过测试结果可以发现，使用位图索引数据库存储网包的索引信息无论在处理时间上还是在查询性能方面都有较高的效率。

3.3.4　TIFAflow 系统[15][16]

TIFAflow 基于 TM 实现的网包信息记录，网络流表维护和网络流量采集及存储的相关模块，并重新设计了 TM 系统的索引部分。索引设计为记录网包在网流文件中的偏移量，查询时根据索引将网包从文件中的特定位置提取出来。通过实验对比了利用通用关系数据库和利用位图索引数据库来存储索引信息两种方案的性能，证明了后者较前者更适合存储索引数据。接着，为了进一步优化系统的存储和查询效率，本节提出了流级的存储和索引设计及实现方案，并利用 FastBit[13] 这一位图索引数据库存储索引信息。实验证明，基于流的改进方案，在存储和查询两个方面都具有更优秀的性能。

1. 设计目标

TIFAflow 系统的设计目标如表 3.6 所示。

表 3.6 TIFAflow 系统的设计目标

系统名称	流级网流归档查询系统
功能简述	对网络中的流量进行采集、存储,并提供方便快速的查询接口
硬件平台	通用 x86 架构的多核处理平台
软件平台	UNIX/Linux 系统
性能指标	在 100Mbps～1Gbps 满负载网络下能实现系统功能并保存数据 30 天
系统输入	可以监听存在的网卡,也可以从流量文件批量读取
系统输出	将捕获的网包归档存储为一定大小的流量文件
查询输入	输入为网流的五元组信息和时间区间,以及查询结果的文件名
查询输出	输出为匹配输入条件的所有网包聚合成的流量文件

2. 系统架构

本节针对表 3.6 中的设计目标,参考 TM 系统,对系统采用采取功能专一化的设计,提出了系统的架构和各个模块的设计。

1) 系统架构图

TIFAflow 系统架构如图 3.21 所示,系统分为两个部分,分别实现网包的存储索引和查询的功能。在现在的设计中,这两个功能相互独立,并且每个功能都是以单线程的方式实现。且每一个功能在设计上都力图最简化,这样可以使得系统各个模块的处理效率更好把握。

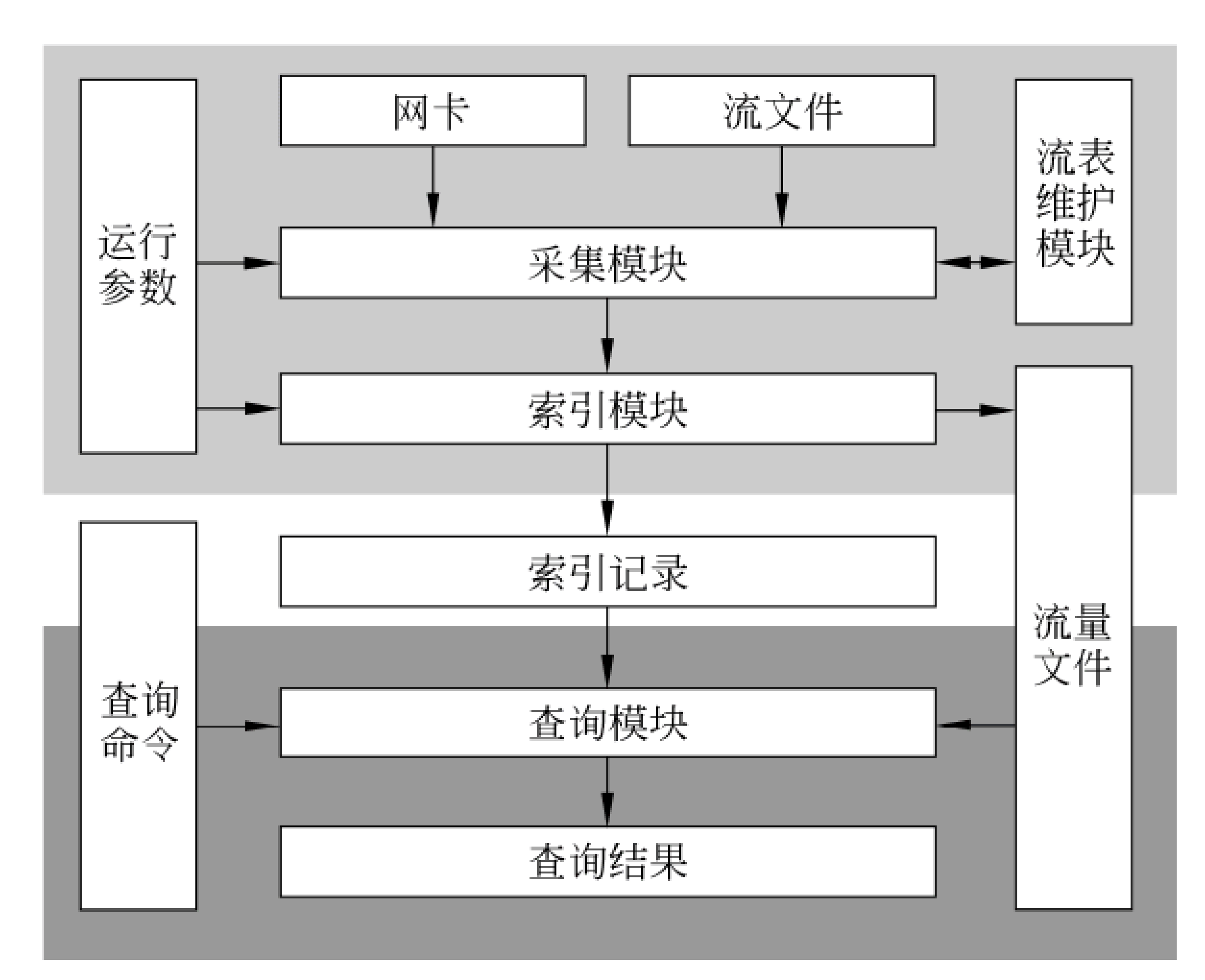


图 3.21 TIFAflow 系统架构

从系统架构图中可以看到，系统的主要模块有采集模块、流表维护模块、索引模块和查询模块。在下面两个小节将说明各个模块的作用和对外接口。

在以后的工作中，这两个部分会在以后的工作中合并在一个进程中以不同线程的方式实现。同时在以后的工作中，查询模块应支持并行的查询，存储和索引模块也将采用多线程的方案，加速处理效率。

2）模块说明

（1）采集模块。

采集模块控制整个系统的输入，根据用户输入的参数，判断是从网卡上读取网包，还是从流量文件中读取网包。采集模块每收到一个网包，将网包交给流表进行处理，流表更新完信息后，采集模块会根据流表模块处理的结果，决定是否将网包交给流索引模块进行处理：如果流表返回的信息是此网包应该被舍弃，则抛弃此网包；否则，将网包传递给流索引模块。

（2）流表维护模块。

流表维护模块维护了当前网络中的所有流信息，一个网包到来时，首先查其对应的网流是否在流表中有记录，如果没有记录，则在流表中新建一条。该记录中记录了流的特征和一些统计信息，如已经处理过的总字节数，流开始时间和属于该流的最后一个网包的到来时间。流表所占用的内存大小应该保持在一定范围之内，因此流表需要定时删除一些过期流的记录，以保证流表不至于太大。

（3）索引模块。

索引模块将到来的包保存到一个流量文件中，同时记录下该包的索引信息。索引模块要维护所有的流量文件，控制每个流量文件的大小。当某个流量文件超过预先设定的大小时，新建一个文件进行写入操作。

（4）查询模块。

查询模块负责解析用户的查询请求。将用户的查询请求转化成自己的查询数据结构，再将此数据交付给索引查询引擎，索引查询引擎在已经建立的索引记录中搜索出相关的索引记录，查询模块利用这些索引记录中的信息从相应流量文件中查找和抽取相应的网包，并汇聚成一个文件返回给用户。

3. 网流获取与存储

1）网包获取

Libpcap 库是 UNIX/Linux 下的一个 C 语言函数库。该函数库提供了捕获和处理包的一些函数。Libpcap 被广泛应用于各类网络监控和分析软件中。

Libpcap 中的一个包的描述使用 struct pcap_pkthdr 进行描述，该结构定义如表 3.7 所示。

表 3.7 Libpcap 库中的 struct pcap_pkthdr 定义

类 型 名	struct pcappkthdr
变量	struct timeval ts； //包的时间戳 bpf_u_int32 caplen； //捕获的包长 bpf_u_int32 len； //网包实际长度

对每一个捕获的包，libpcap 函数库会给出一个网包的描述和指向包头的指针。

2）Libpcap 网包获取方式

Libpcap 库根据用户在运行前输入的参数，提供 3 种方式的网包读取。第一种是直接从网卡抓包，第二种是从某个网流文件中读取网包，第 3 种是从一组网流文件中读取网包。

从网卡抓包需要用户指定监听的网络设备，利用 pcap_open_live 函数接口打开相应的包捕获对象。从某个网流文件中读取包取要用户指网流文件的文件名，利用 pcap_open_offline 获取相应的包捕获对象。从一组网流文件中读取网包时，需要用户指定一个文件，该文件中每一行是一个网流文件的文件名，程序将连续地遍历这些文件中的网包。

4. 流级别的存储和索引

1）系统瓶颈分析

使用位图索引数据库，使得索引的建立和查找时间都大大减少，但是产生的索引文件大小相对较大，不能忽视。在前文所述的方案下，每一个网包都会生成一条索引，不经压缩，一条索引占空间 32b，假设平均包长为 300b，则索引产生的附加存储消耗是 10%左右。另一方面，每一个网包产生一个索引的方案使得索引的数目与网包数相等，对这些索引进行储存和查找是系统的瓶颈所在。如果能减少索引条目的数量，将同时减小建立索引和查找时消耗的时间，提高系统的处理性能。

2）流级别的存储

如果要减少索引数量，却又不想在提取满足搜索条件的网包时，丧失已有设计带来的高效率，就必须利用更少的索引条数存储同样多的位置信息。多条索引的信息想要概括在一条索引中，就必须将信息以另一种方式保存。一个好的想法是让一个流中的包在空间上存储在一起，这样减小索引条数后，那些被放弃的索引的信息蕴藏在了网包的相对位置信息中。一个流存储在一起，使用一条索引标记这个流在所处的文件位置，可以很方便地对一个流中的数据进行遍历，并不会降低系统的性能。同时人们可以做如下计算：若一个流包含 n 个包，则 n 条记录需要被索引；采用流级别的存储，这 n 个网包只会产生一条索引。这样将大大减少所要存储的索引条数，使得数据库的负担更轻。因此，采用流存储的方案，会加快系统处理网包建立索引的处理效率，降低索引存储带来的空间消耗。

3）实现方案

要实现流级别的存储，需要将属于同一个流的网包汇聚在一起，一并存入文件。采用的方法是在内存中利用缓存的方法，将数据一个流的网包存在一起。利用已有的流表，在存储流信息的 Connection 对象中新增一个链表，链表中的每一个节点都是一个网包。

具体流程如下：存储时，系统捕获一个网包后并不马上将该网包存入网流文件并生成索引，而是将其添加到对应的流信息中网包链表的尾端（超过截短界限则丢弃）。当系统清除流表中的超时流信息时，再将要清除的流信息中的网包链表中的所有网包顺序存入 pcap 文件，并生成一条索引，放入数据库中。

索引格式与第 5 章所提方案的格式相似，仅仅需要把时间戳 time 改成时间区间 time_start 与 time_end，并加入一列 flowlen，分别表示流的开始时间与结束时间以及 pcap 文件中该流所占字节数。改动后的索引格式定义如表 3.8 所示。

表 3.8　流级别方案的索引格式

域	类　型	说　明
sip1	byte	源 IP 地址的第一个字节
sip2	byte	源 IP 地址的第二个字节
sip3	byte	源 IP 地址的第三个字节
sip4	byte	源 IP 地址的第四个字节
dip1	byte	目标 IP 地址的第一个字节
dip2	byte	目标 IP 地址的第二个字节
dip3	byte	目标 IP 地址的第三个字节
dip4	byte	目标 IP 地址的第四个字节
sport	short	源端口
dport	short	目标端口
time_start	double	流开始时间戳
time_end	double	流结束时间
fileno	int	pcap 文件编号
offset	int	包在 pcap 文件中的偏移量
flowlen	int	流在文件中所占的字节数
protocol	int	传输层协议（TCP/UDP）

查找时，在索引中查找匹配查找目标的索引，得到相应的流存储的文件和偏移量以及

所占字节数。在相应文件位置把整个流的数据读出。

5. 系统性能

同样，我们测试新的方案完成对 TraceA 的存储与索引需要的时间，并与 TM 所需时间以及 TIFA 的 Fastbit 索引存储方案进行对比，得到图 3.22。从图 3.22 中可以明显地发现，采用流级别的存储方案后系统的性能大大提高，比不使用流存储的方案提速 752s，约 16.6%。

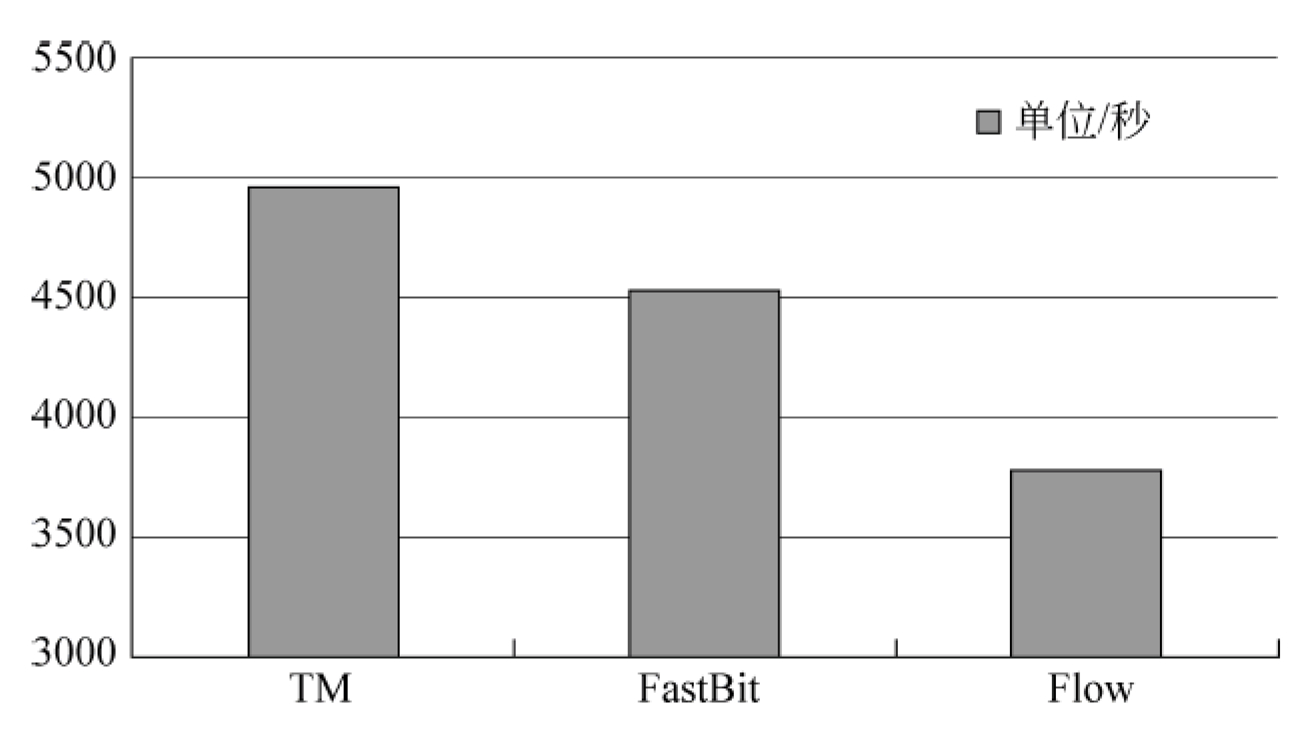


图 3.22 3 种方案的处理时间对比

对比 3 种方案所生成的索引文件占用的空间可得图 3.23。可以看出，使用流级别的存储和索引使得索引大小降低了一个数量级。

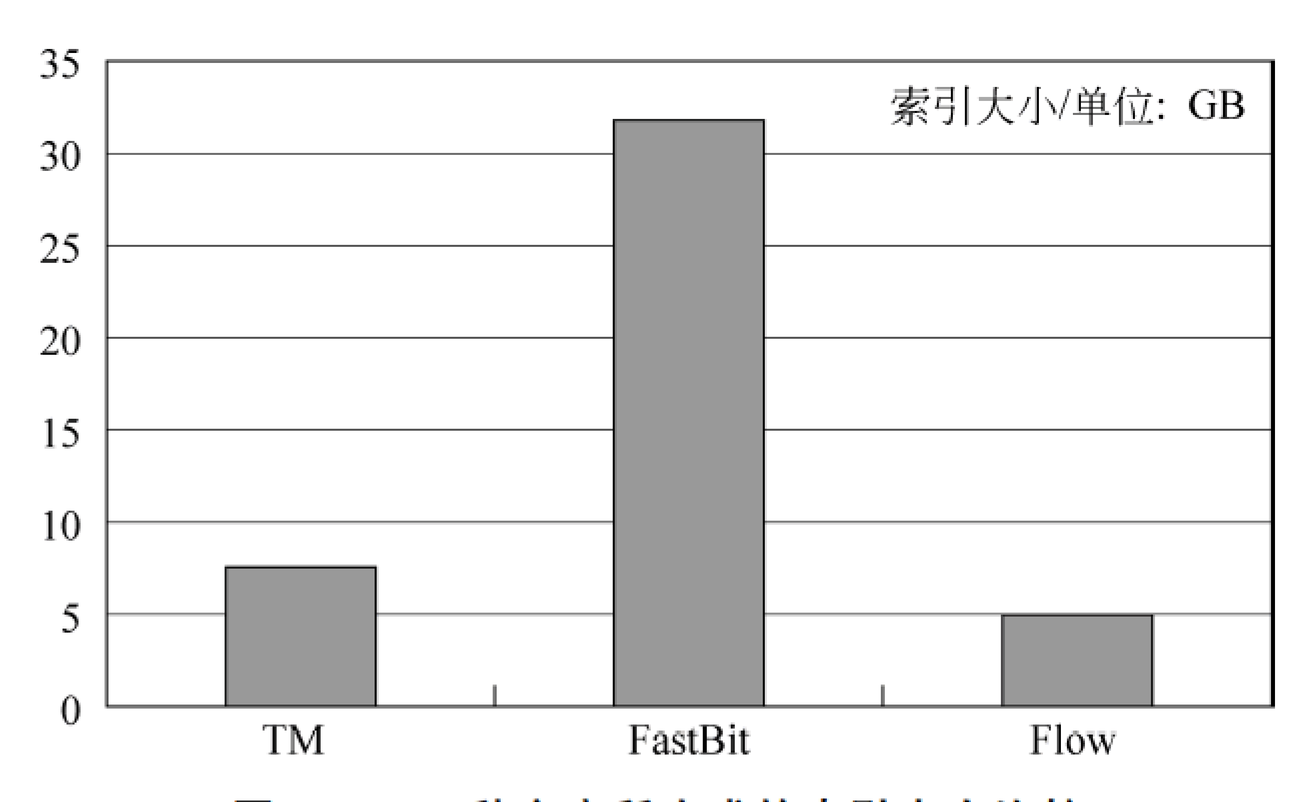


图 3.23 3 种方案所生成的索引大小比较

可以看出，使用流级别的存储和索引方案，能够提高系统存储和建立索引的效率，同时大大降低索引的占用空间。

通过流存储的方案，对 102GB 的流文件的处理时间下降到了 3782s，换算成链路速度约为 215.8Mbps。本次实验的实际网包数据来自于磁盘上保存的抓包文件，假设磁盘的

读取速度是 100MBps，将 102GB 流文件读入内存累积需要时间 1,020s。也就是说，约 1/3 的处理时间用在磁盘读写上。千兆网络环境下，假定网卡 I/O 无瓶颈，对网包完全记录，所设计的系统至少能达到 300Mbps。测试时关闭了包截短功能，因此相当于对全部网包进行处理。在实际运行时，如开启流截短的功能，按照图 3.10 和图 3.11 的流大小分布，该系统工作在 1Gbps 的网络环境下，并保证丢包率在一定水平之下，是完全可行的。

3.3.5 NET-FLI 流记录压缩与查询系统

1. 流记录的属性

NET-FLI 中记录的每条流的具体格式如图 3.24 所示。

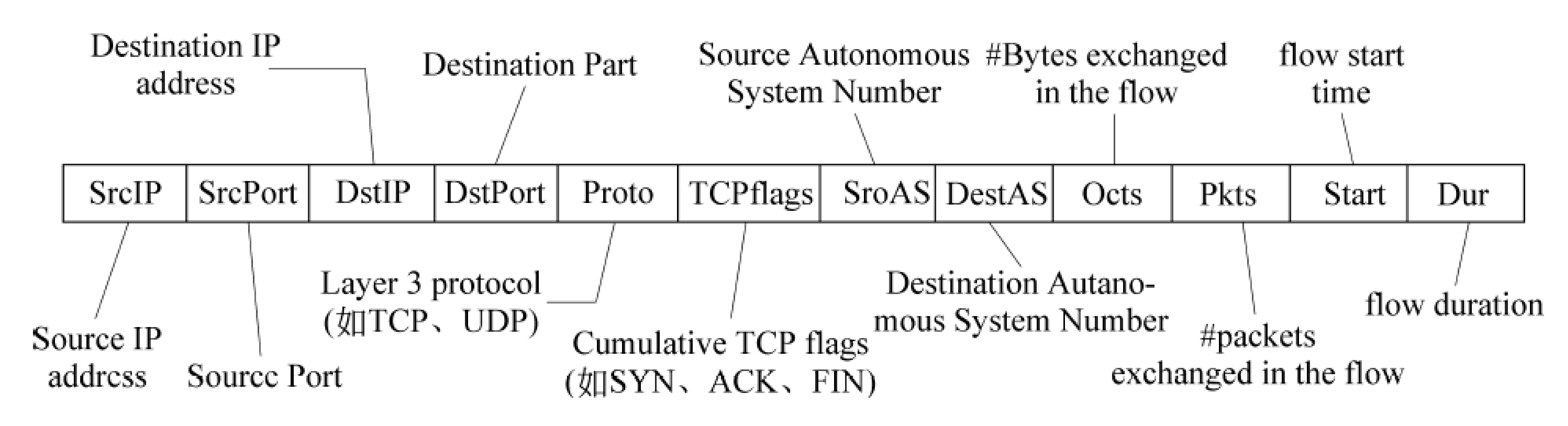


图 3.24 流记录格式

2. NET-FLI 架构

NET-FLI 系统中主要包括 2 个逻辑部分，如图 3.25 所示。

(1) archiving backend 存储后端，压缩流数据(图中 D 部分)。

(2) 压缩索引，对流信息使用 COMPAX 压缩(图中 E 部分)。

这两个组件的任务在 on-the-fly 上被并行地执行，这是我们解决方案高性能的原因。还有一个额外的在线预处理步骤被执行，使用 oLSH 技术对流进行排列，目的是对归档数据和数据索引实现更好的压缩。这个可选的操作可能在一定程度上降低了流处理的速率，但是它能够得到更高压缩的归档数据和索引，最后的响应时间也就会越快。

系统的最后一部分是 query processor(查询处理器)，给定一个搜索查询和使用位图索引结构，通过匹配给定的标准确定历史流，通过解压缩归档数据中相关的部分来恢复它们。

3. 流数据的压缩归档

流记录中可以含有一系列的属性，如图 3.24 所示，到来的流记录会被分包，使用 4000 个流的窗口来处理。窗口的大小并不是随意选择的，它能反映处理数据的数量(12 ×4000)，并且和 L2 缓存相适应。该篇文章的作者曾经也做过更大流记录块的实验，但发现这并不明显地影响压缩速率，所以，这里使用 4000。

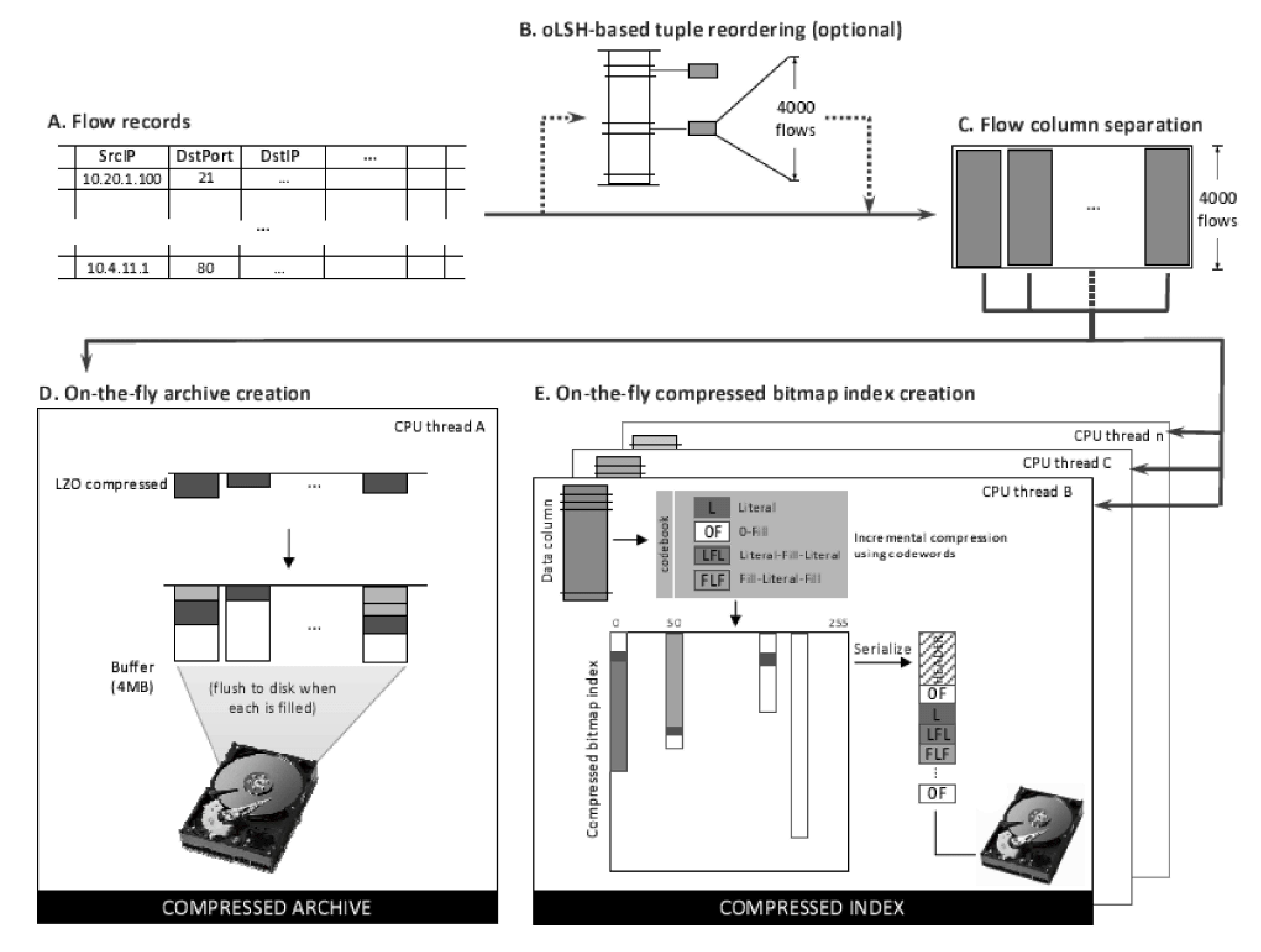


图 3.25 NET-FLI 架构示意图

在后来的阶段中,数据都被看成 columnar (一栏一栏)形式,每个属性都代表一栏,这些都是独立处理的,每栏的长度都是 4000。我们选择 LZO 压缩器,压缩后的数据归档在 on-the-fly 上创建,并存储在 disk 上。压缩后的块存在 buffer 中,buffer 满了才倾倒到 disk 上,默认 buffer 大小为 4MB。

这部分任务主要有分包、重新排序和压缩,是在一个 CPU 系统中执行的。

4. 索引压缩算法 COMPAX+oLSH

该系统使用的索引压缩算法是 COMPAX 的改进版——COMPAX + oLSH,COMPAX 在本章的第 2 节中有介绍,其中 oLSH 方法(online-Locality-Sensitive-Hashing)是一种聚集相似流记录的方法。该改进压缩方式同 COMPAX,但是会对输入码流进行事先处理,将相近的包放在相近的位置,这样对压缩率的提高非常明显。

5. NET-FLI 查询系统

已知:源 IP 为 10.4.0.0/16,目的端口为 137。

查询:所有的目的 IP。

NET-FLI 查询过程示意图如图 3.26 所示。

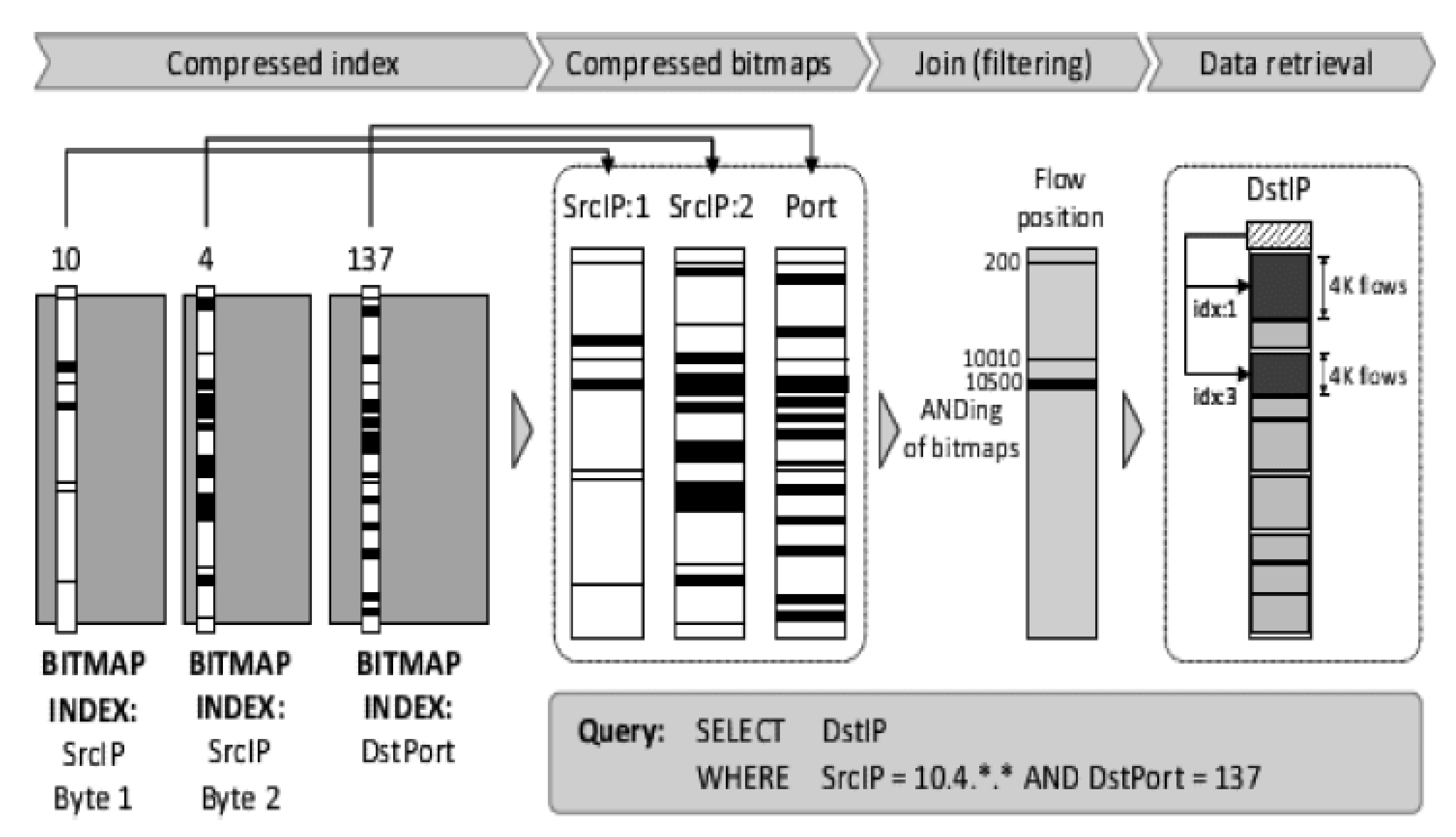


图 3.26　NET-FLI 查询过程示意图[11]

步骤如下。

（1）找出压缩 index 中的 3 个 columns（源 IP 的前 2 个字节为 10、4，目的端口为 137）。

（2）将 3 个 columns 进行与操作。

（3）连接 3 个压缩 columns 的结果有匹配的行{200，10010，10500}。

（4）解压，还原出目的 IP（找到行所对应的块，从头部得到开始位置）。

（5）返回给用户。

3.3.6　Hyperion

Hyperion[8]是一个支持存档、索引、对大量数据流进行联机查询的系统。采用一个对高速存储数据流的写优化流文件系统，并采用分布式 multi-level 的索引结构。Hyperion 是在通用硬件上进行实现，StreamFS 文件系统每秒存储百万个包，在处理 on-line 请求的同时，索引和存储可以达到每秒 200 000 个包。

网络监测一般分为两种，一种是实时监测，即实时捕获和检查包，在检查完之后，丢弃包的头部和负载；另一种网络监测系统除了需要线速地捕获数据，还要有存储、回溯查询和高效检索数据的功能。如今，对存档系统一般有两种可选方式，可以使用数据库，也可以在常规文件系统上自定义建立索引。

像 GigaScope 和 MIND 使用 SQL 接口来实现网络数据的查询。一个检测系统必须在高速情况下接受新的数据，单个 gigabit link 每秒可以产生成千上万个包头和几十 MB 的存储数据，单个检测系统可以记录多个这样的 links。用常规的数据库系统是无法满足这样的要求的。MIND 是基于 p2p 的索引，仅提取和存储 flow level 的信息，而不是 raw

packet header。GigaScope 是一个流数据库,可以支持实时数据的请求,但数据的存储并不在考虑范围之内,它们之间的主要区别如表 3.9 所示。

表 3.9 主要区别

	Event rates	Archive	Index,query	CommodityHW
Streamingquery systems(GigaScope, Bro,Snort)	Yes	No	No	Yes
Peer-to-peer systems (MIND,PIER)	No	Yes	Yes	Yes
Conventional DBMS	No	Yes	Yes	Yes
CoMo	Yes	Yes	No	Yes
Proprietary systems *	?	Yes	Yes	No

可以采用通用的文件系统来存储捕获的包头部,特别是日志文件,并且在这些文件上可以自定义索引进行有效的查询。但是通用的文件系统并不是被设计用来网络监测的,所以吞吐率低的系统可能不适用这种情况。

因为需要对大容量的数据进行存储,所以数据库和常规的文件系统都不能直接地解决存储上的问题。这就需要一个工作在通用硬件上的新存储系统,主要用于解决大容量存储、建立索引、查询等问题。下面将从流文件系统(Stream File System)和分布式索引结构两部分介绍 Hyperion 系统。

1. 流文件系统

流文件系统的设计需求包括以下要点。

(1) 对高速网流的存储不能有信息丢失。

(2) 对整个磁盘的重复存储使用。

(3) 读和写可以并发。

在流文件系统中,数据块(block)的大小是固定的,其中包含块头(block header)和多个记录(records),块头指定了该块属于哪个网流,以及将块和上一块的记录分离开。每个记录的长度都是可变的,同一个块中的记录是同属于一个流,记录头部(record header)中包含流 ID(stream ID)和记录的长度,并且出于对安全性的考虑,在记录头部中加入一个自我认证的机制,即在记录头部中包含一个头文件的散列值。假设 n 个块组成一个区(zone),那么前 $n-1$ 个块是用来存储记录,最后一个块则用于存储数据块图(block map),将流 ID(stream ID)和各个数据块 block 之间建立映射关系。而流文件系统 Root 就像一个总目录,包含所有的流(stream)的信息,存储了每个流的头尾指针和大小。具体结构如图 3.27 所示。

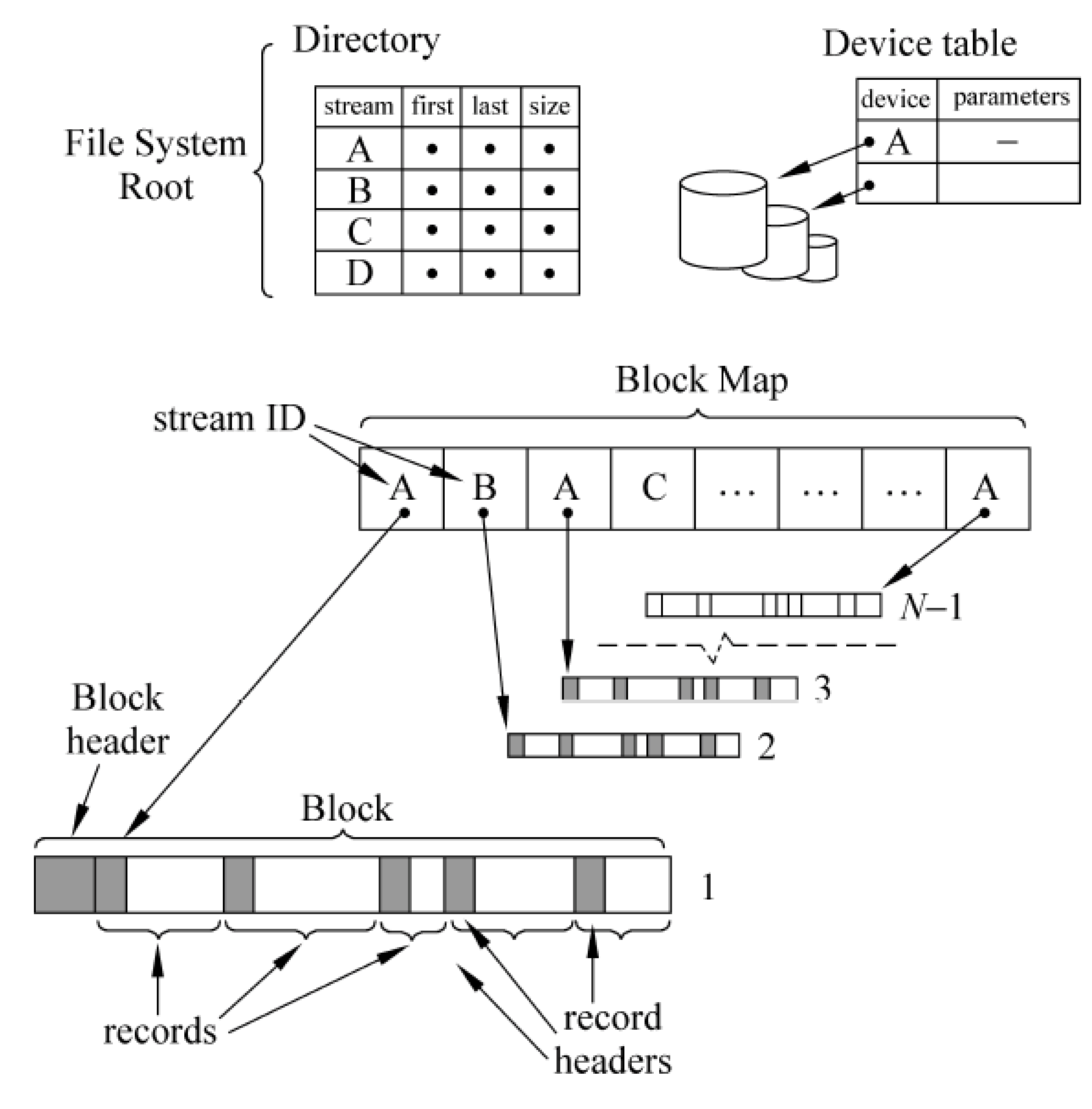


图 3.27　流文件系统的存储结构

2. 分布式索引

Hyperion monitor 需要维护一个可以高效回溯查询，并且能够被快速建立的索引。为了能达到索引每个链路上每秒 100 000 条记录的速度，Hyperion 将一个流 stream 分成多个区间 interval，给这些区间 interval 计算一个或多个签名。所以特征索引 signature index 维护的不是 stream 索引，而是区间 interval 的索引。索引中存储的不在是数据，而是一个个的特征 signature。只有在特征 signature 完全匹配了，才进行数据的恢复。

Hyperion 分布式索引方案如图 3.28 所示。

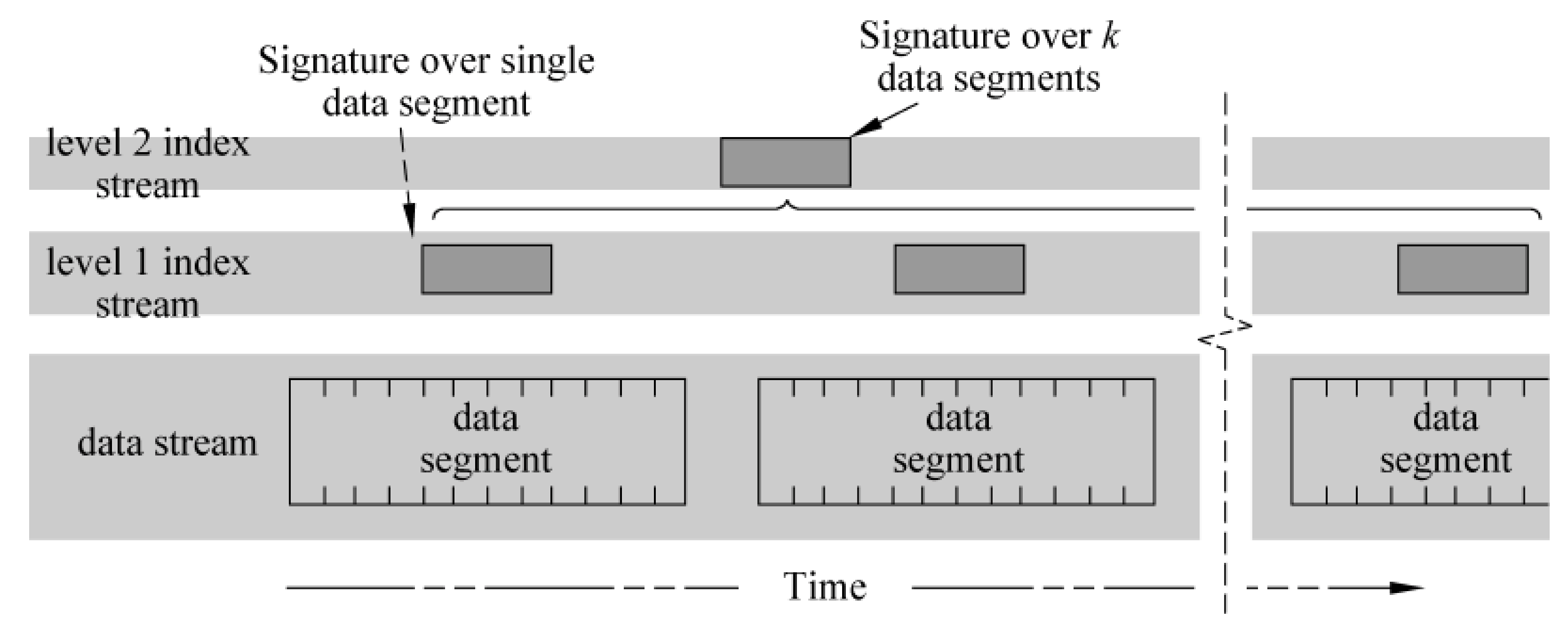


图 3.28　Hyperion 分布式索引方案

Hyperion 采用多级特征索引(multi-level signature index),这里使用 2 层签名索引结构,第一层是每个区间 interval 都会计算得到特征 signature,第二层则是由 k 个区间 intervals 计算得到特征 signature。每次先查找高层(第二层)的 signature,如果匹配,说明在该 k 个区间(intervals)中,再查找 level1 的 k 个特征(signatures);如果没有找到匹配,就说明在这 k 个特征 interval 中是没有的。所以这种特征索引(signature index)方法不会出现漏报情况,但会出现误报(即检查的 interval 并不是需要找的 interval)。

Hyperion 采用的是分布式索引结构,但是这种结构很容易因为对查询请求的回答而造成查询洪泛(query flooding)。在 Hyperion 中每个节点本地存储自己的索引信息,其中每个节点的汇总 summary 索引(top level 的索引信息)也是需要生成的,这里在某个时间段里指定一个节点 A,让其他节点把 top level 的索引信息发送给节点 A,由节点 A 来做匹配,最后在与自己匹配中查找。

3.4 处理平台展望

3.4.1 多核处理平台[17]

多核系统是计算机处理器的一个主流形式,但是,内核网络层并没有被完全利用,这是因为存在的老旧代码、RX 队列的资源竞争以及 OS 系统之间不必要的副本,这就导致了包捕获在适应多核架构时可能会出现性能低下。

现代的多核网络适配器被逻辑上分成了几个 RX/TX 队列,并使用 RSS 的方式平衡流,队列的个数是依赖于网卡的芯片设置和可用的系统 cores(比如 4 核的只能有 4 个队列 per port)。但由于大多数的 OS 都是使用 pre-multi-core 时设计的包轮询机制,当时的网络适配器是只有一个单独的 RX 队列。即使现今的机制可以并行多个线程,但由于还是对一个资源进行竞争,所以没有发挥出并行多核架构的优势。

多核处理平台示意图如图 3.29 所示。

3.4.2 GPU 方法[18]

这里简单介绍一种新方法,就是利用 GPU 来处理对数据索引,达到解放 CPU 的目的。

从图 3.30 上很容易看出数据从 CPU 上复制到 GPU,GPU 上通过一些列的处理生成 metadata 返回到 CPU 上,metadata 由 2 个 parallel array 组成,keys 和 offsets,keys 是分类和存储 m 个不同的 keys(m 代表出入的数据不同值的个数),offsets 是给每个 key 存储在 index 中的偏移量。

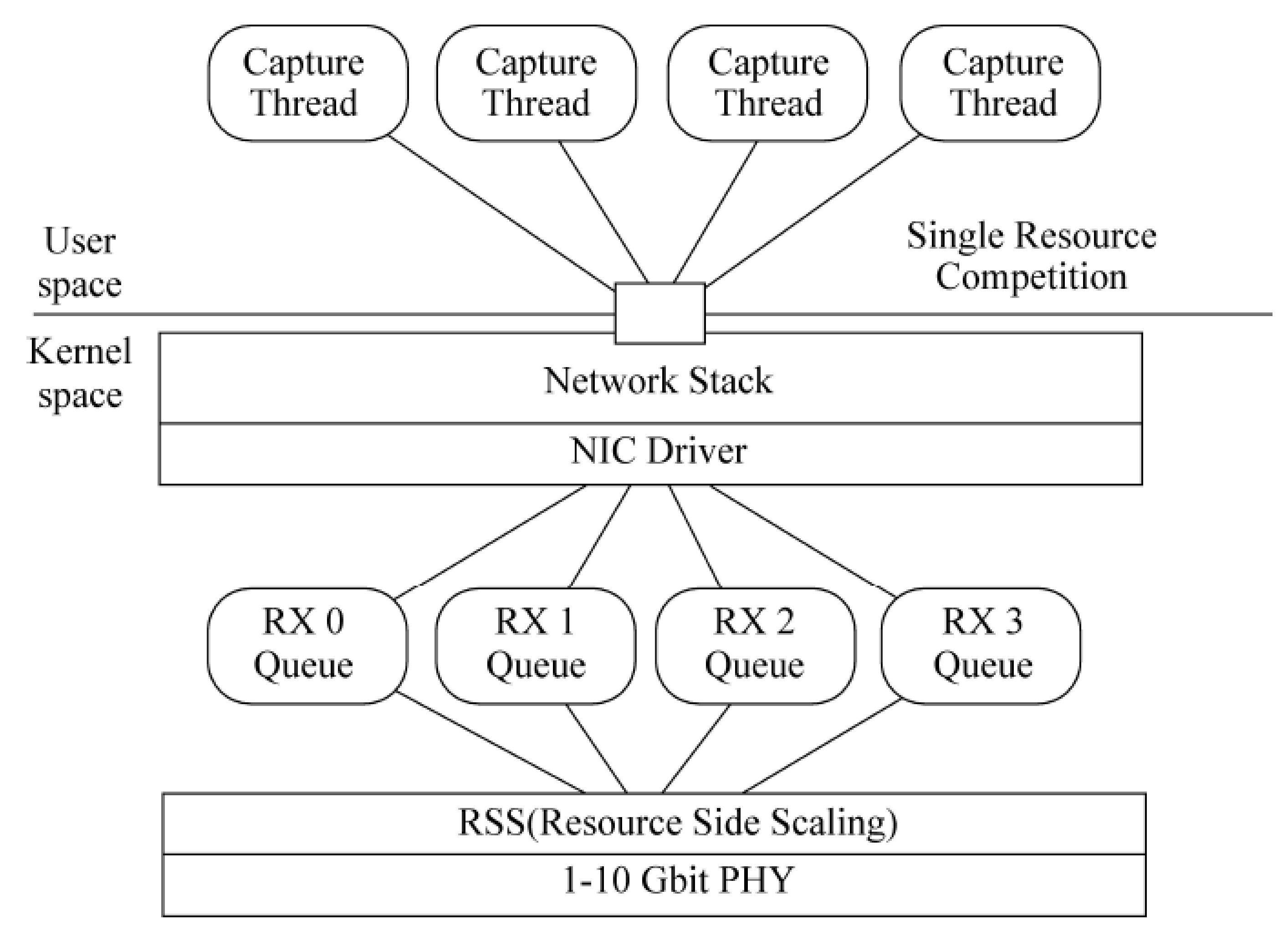


图 3.29　多核处理平台示意图

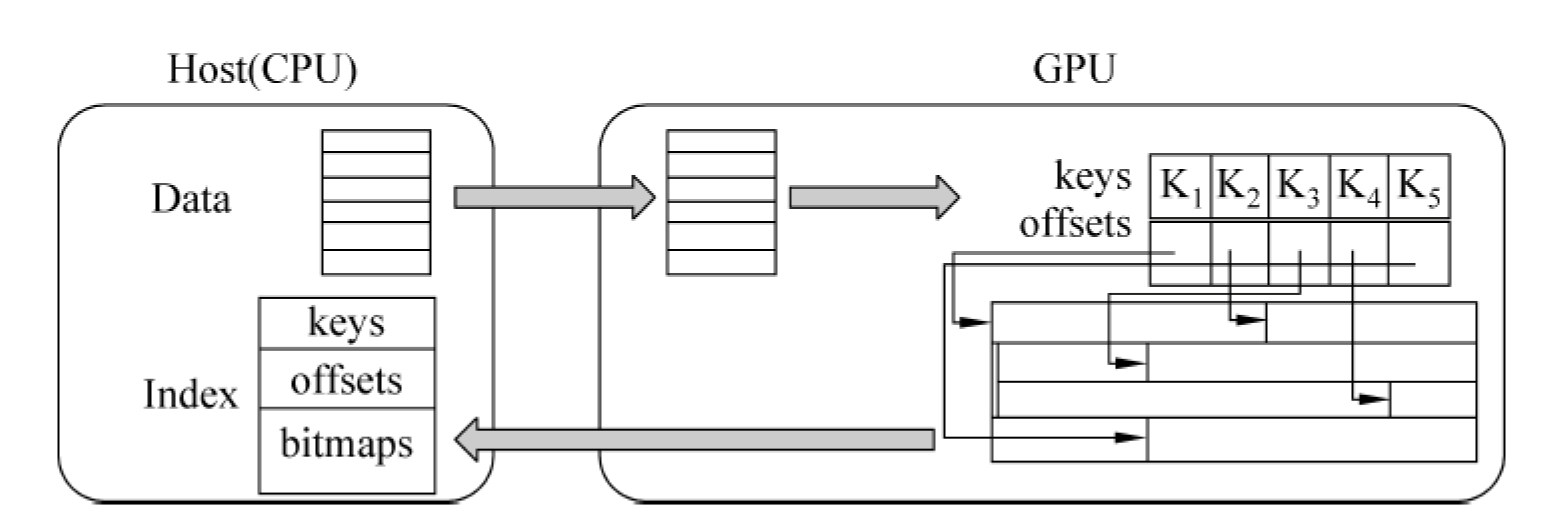


图 3.30　GPU 处理数据索引示意图

3.5　小　　结

本章介绍了通用网络流量归档与查询系统中的基础知识和网包处理过程中的相关基础知识，以及线速处理的相应技术手段，主要包括网包获取的 Linux 内核 NAPI 技术、libpcap 库、netmap 高速网包处理框架、PF_RING、网流获取与处理的 scap 库。同时，介绍了网包的索引压缩算法，包括 WAH、PLWAH 和 COMPAX 算法等关键技术。接着，介绍了单点的网流归档与查询系统的具体实现，包括基于 MySQL 关系型数据库的存储查询、TM 存储与查询系统实现、TIFA 系统实现、TIFAflow 系统实现和 NET-FLi 系统实现。最后，介绍了面向网流归档与查询系统应用的通用多核 multi-core 平台与 GPU 平台方案。

参 考 文 献

[1] Cisco Visual Networking Index Forecast (2011—2016).
http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html.

[2] libpcap，Network Research Group，http://www.tcpdump.org/.

[3] A. Tuner，M. Bing. tcpreplay. http://tcpreplay.sourceforge.net/.

[4] The Tcpdump Group. tcpdump. http://www.tcpdump.org/.

[5] Fusco，Francesco，Xenofontas Dimitropoulos，Michail Vlachos，Luca Deri. PcapIndex：an index for network packet traces with legacy compatibility. In proceeding of ACM SIGCOMM Computer Communication Review 42，2012，no. 1：47～53.

[6] Deri，Luca. Improving passive packet capture：Beyond device polling. In Proceedings of SANE，2004，vol. 2004，pp. 85～93.

[7] Rizzo，Luigi. netmap：a novel framework for fast packet I/O. In Proceedings of the 2012 USENIX conference on Annual Technical Conference. USENIX Association，2012.

[8] Desnoyers P，Shenoy P J. Hyperion：High Volume Stream Archival for Retrospective Querying [C]. USENIX Annual Technical Conference. 2007.

[9] K. Wu，E. J. Otoo，A. Shoshani. Optimizing bitmap indices with efficient compression. ACM Transactions of Database Systems. 2006. 31：1～38.

[10] F. Deliège，T. B. Pedersen. Position list word aligned hybrid：optimizing space and performance for compressed bitmaps. In Proceeding of the 13th International Conference on Extending Database Technology，2010.

[11] F. Fusco，M. Stoecklin，M. Vlachos，NET-FLi：On-the-fly Compression，Archiving and Indexing of Streaming Network Traffic. In：Proceedings of the International Conference on Very Large DataBases (VLDB). 2010.

[12] KORNEXL，S.，PAXSON and others，Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic (Short Paper). In Proc. ACM IMC (2005).

[13] G. Maier，R. Sommer，H. Dreger，A. Feldmann，V. Paxson，and F. Schneider. Enriching Network Security Analysis with Time Travel. In Proc. ACM SIGCOMM，Seattle，WA，Aug. 2008.

[14] Li J，Ding S，Xu M，et al. TIFA：enabling real-time querying and storage of massive stream data [C]. Networking and Distributed Computing (ICNDC)，2011 Second International Conference on. IEEE，2011.

[15] Z. Chen et al.，TIFAflow：enhancing traffic archiving system with flow granularity for forensic analysis in network security. Tsinghua Science and Technology，vol. 18，No. 4，2013.

[16] Z. Chen，Xi Shi，Ling-Yun Ruan，Feng Xie，Jun Li，High Speed Traffic Archiving System for Flow Granularity Storage and Querying，ICCCN 2012 workshop on PMECT，2012.

[17] Fusco, Francesco, and Luca Deri. High speed network traffic analysis with commodity multi-core systems. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, 2010.

[18] Fusco, Francesco, Michail Vlachos, Xenofontas Dimitropoulos, Luca Deri. Indexing million of packets per second using GPUs. In Proc. of the 13th ACM SIGCOMM Conference on Internet Measurement, 2013.

[19] 陈震,温禹豪,马戈,曹军威. 一种压缩比特位图索引的方法. 2013.

[20] 陈震,温禹豪,马戈,曹军威. 一种新的压缩比特位图索引的方法. 2013.

[21] 陈震,刘洪健,曹军威,基于倒排序表的网流索引检索与压缩的方法. 2013.

[22] Yuhao Wen et al., SECOMPAX- a bitmap index compression algorithm, in Submission.

[23] Yuhao Wen et al., ICX- a bitmap index compression algorithm for Internet Traffic Archival, in Submission.

[24] 陈震,温禹豪,曹军威. 一种最大步进携带比特位图编码方法,2014.

第 4 章

互联网流量大数据存储

4.1 流量大数据-移动互联网增长背景

据中国联通内部统计，在过去 5 年中移动用户流量的复合增长率为 135％，3G 月均流量增长最高，其中以 iPhone 用户尤其突出，如图 4.1～图 4.4 所示。

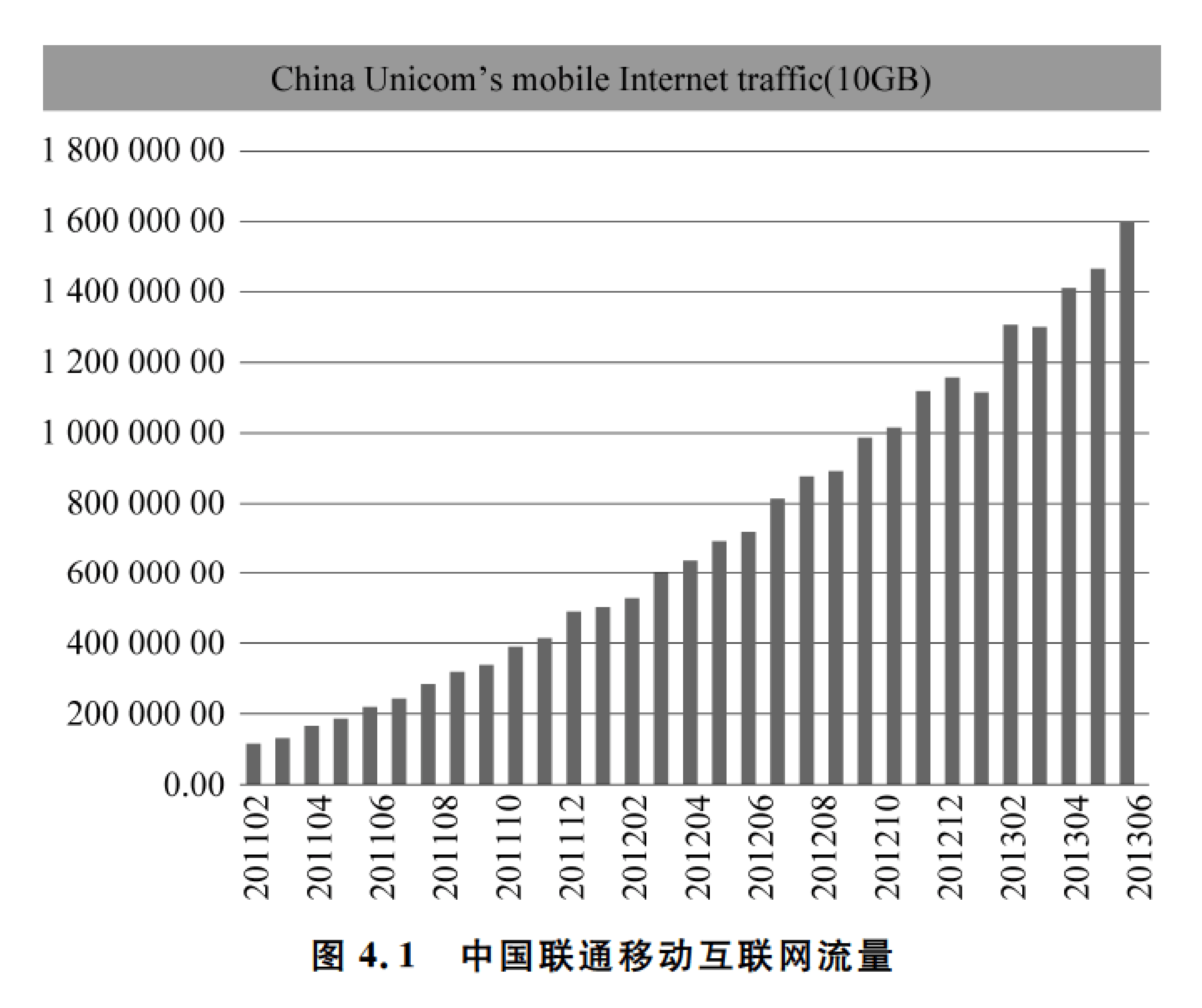


图 4.1 中国联通移动互联网流量

流量增长原因如下。

(1) 联通 3G 无线网络的普及。

随着移动通信网络的发展，移动设备能够随时随地访问任何内容。移动访问速度从原来的几百 KB 上升到 MB，未来 4G、5G 时代的移动访问速度或将达至百 MB。

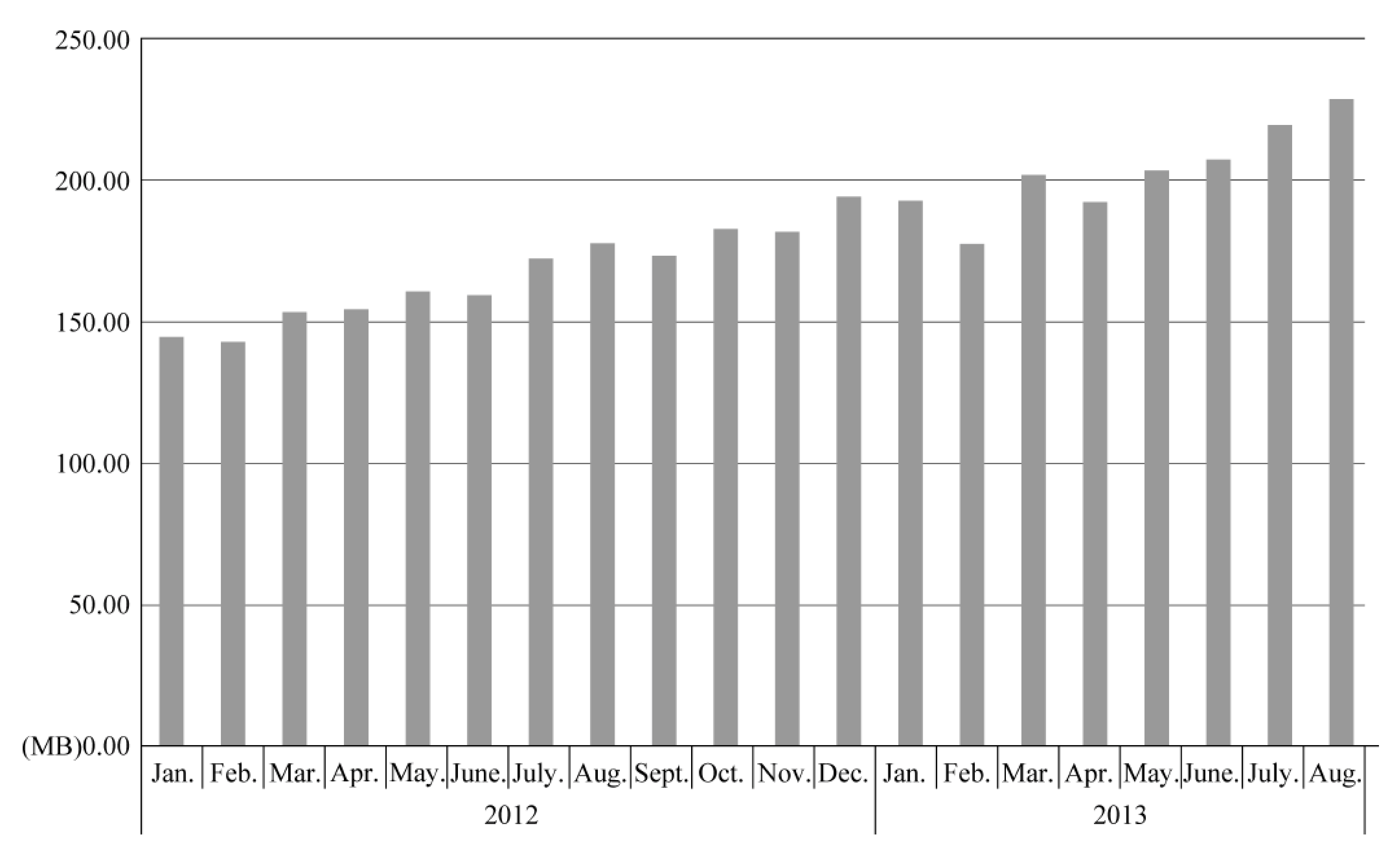


图 4.2　移动户均移动 3G 流量(2012—2013)

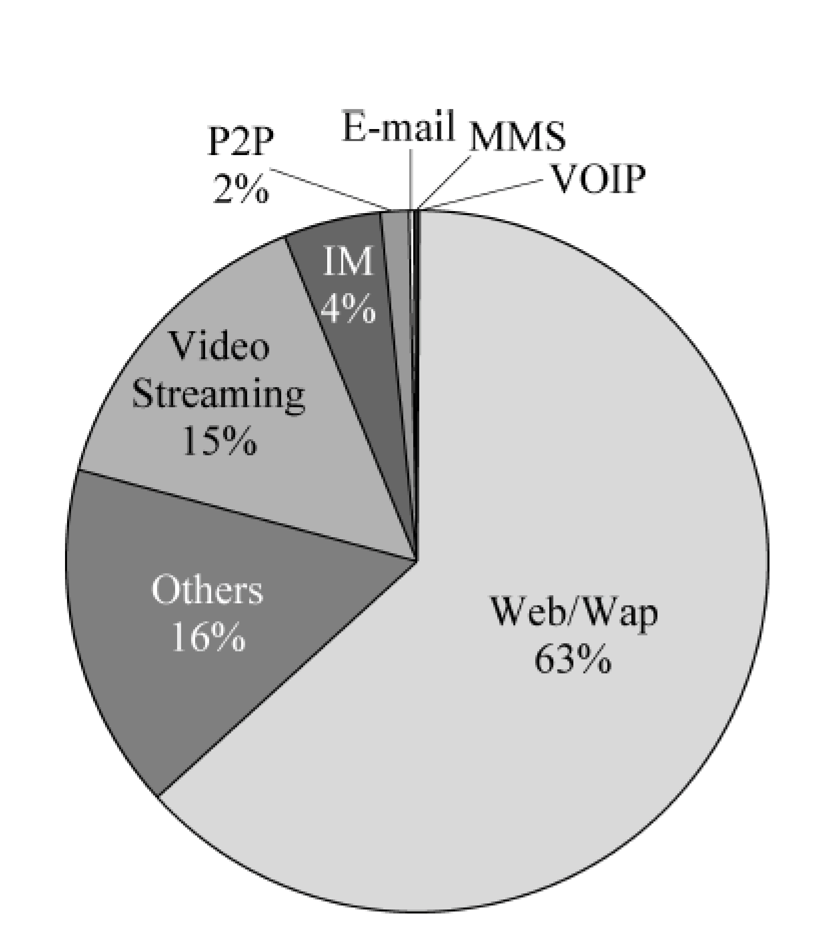


图 4.3　流量类型分布(视频、网页、语音等)

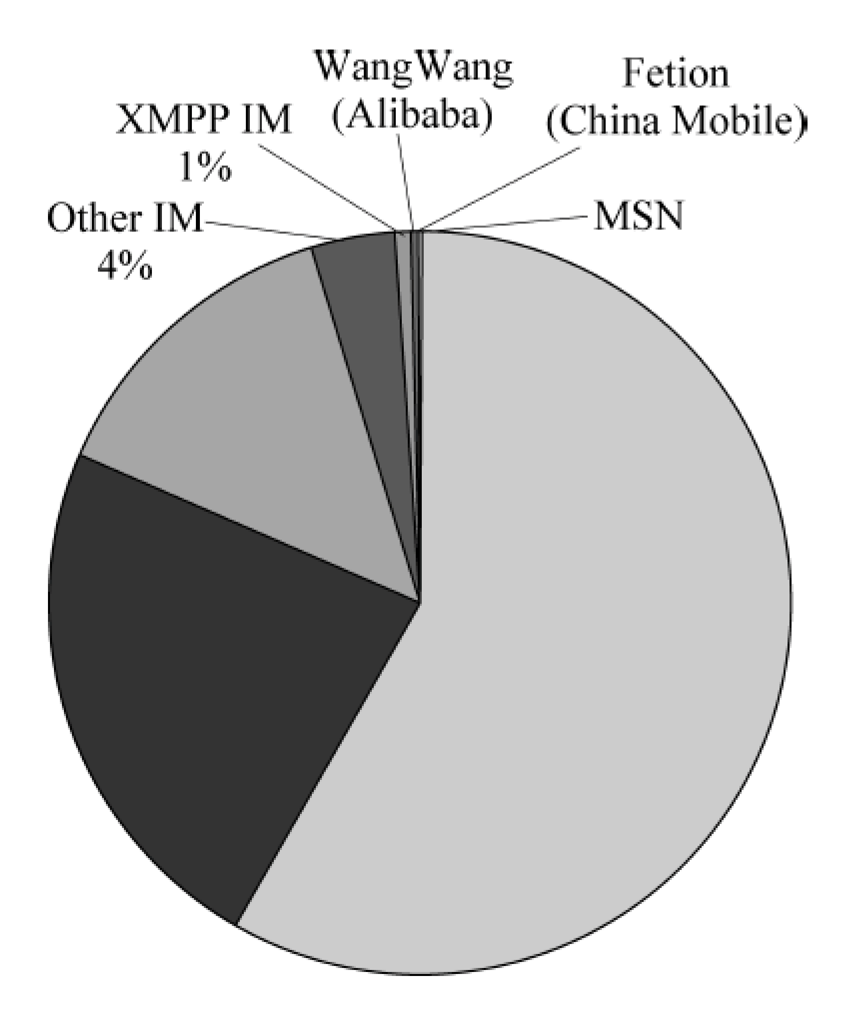


图 4.4　即时通信流量分布

(2) 终端设备能力提高。

随着移动设备终端的快速发展,iPhone 处理器从 32 位架构进化到当前 iPhone 5S 版本的 64 位架构,仅用了 6 年的时间(2007—2013)。而在桌面计算上,Intel 处理器从 32

位到 64 位的发展整整持续了 20 年(1985—2004)。手机的存储容量也随着存储设备的价格降低而不断增加。

(3) 丰富的互联网内容向移动互联网转移。

近年来,移动互联网的内容愈加丰富,手机游戏、手机视频等呈现爆炸式增长。目前移动互联网最活跃的应用是移动游戏。

按照 IDG Gartner 的定义,大数据符合 4 个特征,即数据体量巨大(Volume)、数据类型繁多(Variety)、价值密度低(Value)、处理速度快(Velocity)。随着移动互联网大幕的开启,传统移动运营商的商业模型也在不断革新,运营商的流量大数据经营时代已经到来。

4.2 流量大数据-采集、获取与归集

第 3 章详细介绍了流量采集的基本技术,以及流量归档查询系统的设计和实现。在 TM 技术基础上,我们开发了 TIFAflow 流量归档查询系统[4-5] 和 TIFA 归档查询系统[6],TIFAflow 系统部署在多个站点,是 CNSMS 协同安全防护体系[7]的重要组成部分。网络流量的采集取证分析,在 Botnet 网络压制[8][9]、反垃圾邮件和钓鱼攻击等网络欺诈[10]中,具有重要的作用。同时,也可以通过流量管理设备,对网络进行实时管控[1]。

中国联通总部维护着所有移动互联网用户的 CDR(Call Data Record or Charge Data Record),每天维护的 CDR 高达 800 亿条。在每个省的移动互联网出口设备 GGSN 的 Gn 点都部署有 DPI 采集设备,全国有大约 100 多个 GGSN 的采集点,DPI 采集设备采集所有的 IP 数据包,并进行适当的合并。合并的原则是既不能丢失有效的用户行为数据信息,又要讲究合并的效率,以减少无效的数据量。生成的每个文件都小于 200MB,而这样的一个文件在 5min 内即可生成。每 200MB 文件中包含的记录大约有 70 万条,如图 4.5 所示。

获取的流量数据包括如下。

(1) 用户访问了什么样的内容和服务。

(2) 移动终端的分布。

(3) 移动人群的位置信息和人流分布状况。

(4) 用户偏好。

图 4.5 显示了运营商流量大数据的采集与检测系统,其中 SGSN 和 GGSN 对流量进行分光映像,进行流量记录流量档案系统(traffic archival system)。

图 4.6 是联通大数据平台的主体架构,该平台从 2009 年 10 月上线,每月记录超过 2 万亿条(2×10^{13}),每月数据量超过 525TB,数据总量已达 5PB。整体入库速度达到 139 万条/秒,万亿条记录规模的大表中记录检索时间小于 100ms。

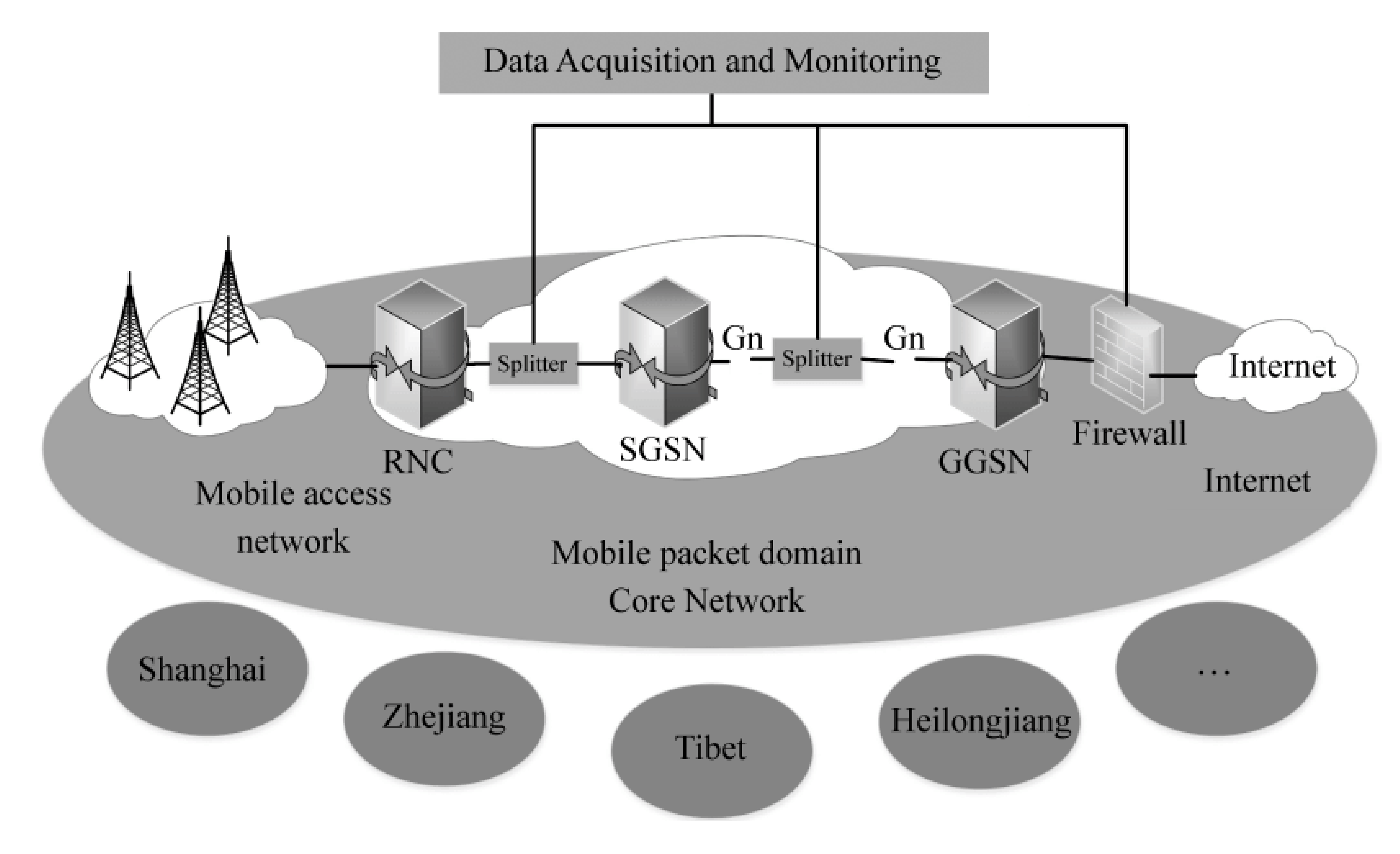


图 4.5　数据采集与检测系统

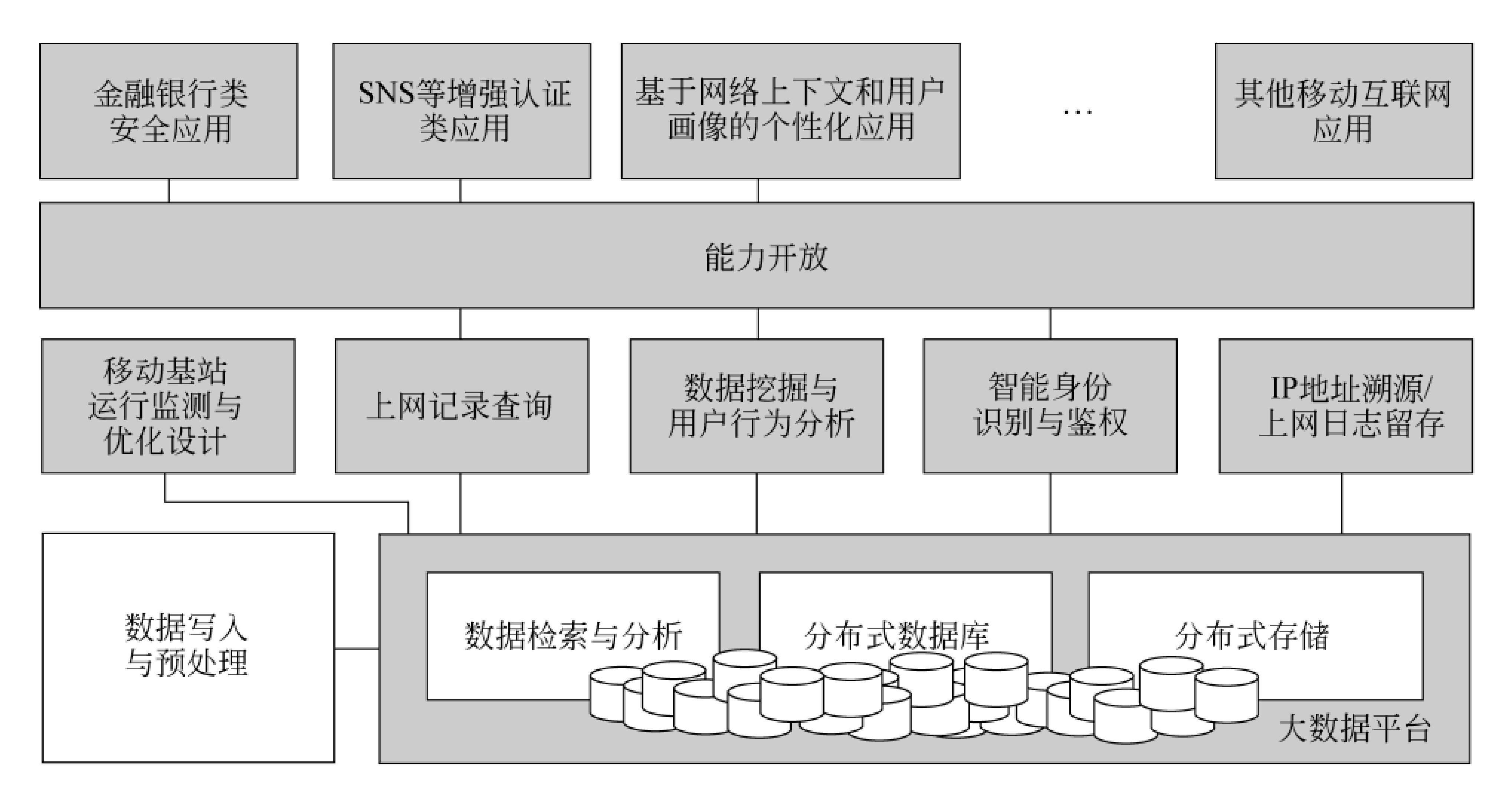


图 4.6　联通大数据分析与挖掘平台

采集部分支持所有的业务类型解析，统一按照集中化上网记录查询及分析系统的私有协议封装后进行传输，各个字段之间以竖线(|)作为分隔符。文件生成周期为 5min(默认值，可以设置)，单个文件大小设置为 200MB，在两个指标同时设置时，单一指标到达阈值则结束文件。每个时间周期内，文件大小超过设定的大小阈值，则保存多个文件，同一周期内的文件通过[nnnnn]序号区分。具体格式见附录 A。

文件通过 FTP 上传到北京的 24 台 FTP 服务器，规模较小的省，两个省共用一台 FTP 服务器，有些大省则需要两台 FTP 服务器。为了减少传输的带宽，这些文件从省里上传到北京时都采用了 bzip2 压缩算法进行了压缩。位于 FTP 服务器集群上运行着入库程序，读取通过省内 FTP 上传的文件，经过解压缩后，通过 HBase 提供的 Native Java API 写入 HBase 数据库，HBase 数据库按月生成一张上网记录表，该表以用户的手机号码作为 ROW KEY，用户通过手机号码可以快速地检索自己的上网记录。

4.3　流量大数据-平台架构及系统实现

4.3.1　Hadoop 集群

目前，联通大数据平台中集群可用存储是 1.9PB，已使用存储是 1.43PB，存储使用率为 74.1%。2013 年 9 月底使用存储 1.63PB，使用率达到 85%左右。包括如下。

（1）集群规划 188 个节点，目前实际使用 169 个节点（10 个管理节点＋159 个数据节点）。

（2）NameNode 3 台，采用 HA 部署。硬件配置为每台两个 4 核 CPU，96GB 内存。NameNode 和 jobtracker 部署在同一台服务器。另有 SecondaryNameNode 和 Standby NamNnode。

（3）Zookeeper 和 HMaster 7 台，硬件为每台 2 个四核 CPU，24GB 内存。每台机器上部署 Zookeeper 服务和 HMaster 服务。为 HDFS 和 HBase 服务。

（4）数据节点 159 台，为 2 个四核 CPU，48GB 内存。部署启动了 datanode、regionserver 和 tasktracker 服务。数据节点为 14 块 1TB 硬盘，做了 14 个 raid 0，挂载到系统中。每个节点实际可用存储 12TB。

剩余服务器，作为分析集群使用。当负载过高时，这些服务器也可以加入到大的生产集群中。

HDFS 内部的块选取策略的基本思想很简单，在每一个 DFSClient 以及 DataNode 处维护一个可达的 DataNode 列表，每次访问某个 DataNode 后记录访问的延时，如图 4.7 所示。这些信息汇报给 NameNode。这有点像 OSPF 路由算法，每一个路由器维护一个邻接图，图中每一条边都有访问延时信息。

客户端 DFSClient 在选择数据块进行访问时，首先问 NameNode 哪些 DataNode 有这个数据块（block）。NameNode 会根据已有的延时信息返回 DataNode 的列表，按照延时排序，访问速度最快的 DataNode 排在最前面。DFSClient 收到 DataNode 列表后，如果本地有延时信息记录，则使用本地信息找到访问最快的 DataNode，否则就按照 NameNode 给的 DataNode 列表，去访问第一个 DataNode。

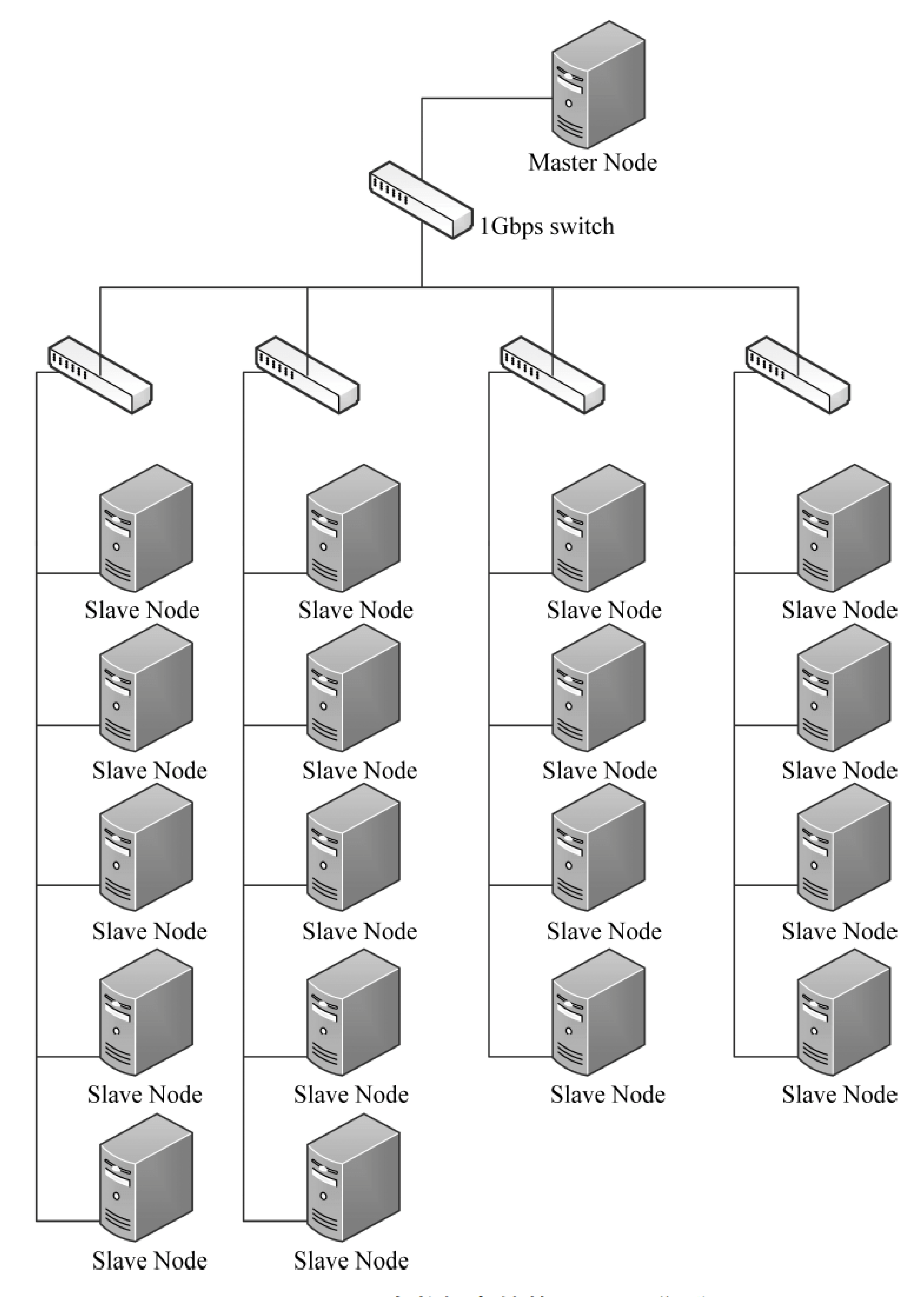


图 4.7　用于大数据存储的 Hadoop 集群

4.3.2　HBase 集群

上网日志详单系统的 HBase 集群规模达到了 188 台机器，裸容量达到 2.5PB，有效容量达 1.25PB，是目前所知电信业内规模最大的 HBase 集群，每天可以处理数十 TB 级别的数据量。相对而言，根据公开资料，Facebook 的单个 HBase 集群为 100 台服务器，淘宝的 HBase 集群有 10 个，共 300 台服务器。

HBase 大规模分布式系统的主要技术难点如下。

1. 高并发数据导入

联通全国范围的实时上网记录需要在短时间内导入到系统中，要求系统的峰值处理能力在每秒 100 万条上网记录以上。为了达到如此高的性能，同时又需要允许客服人员和用户能够快速查询和检索数据，采用了下列三项新技术。

（1）采用新型数据库设计思想中的多版本并发控制（MVCC）数据处理模型，每插入一份数据就形成一个新的版本（时间戳），并且与读数据的操作分离。允许高并发的用户同时访问和插入（修改）数据。

（2）现代硬盘的 IOPS（每秒 I/O 次数，通常为 1～100IOPS）较低，但是磁盘带宽（每秒传输数据量）相对富余（100MB/s 左右）。为解决传统数据库由于建立 B＋树索引导入数据越多性能越慢的问题，需要避免频繁进行 I/O 操作。在实现 HBase 系统时，首先用顺序磁盘写代替随机写来创建数据和索引，对数据块进行高效的合并操作；其次，为了减少合并操作对性能的影响，对上网记录这种不需修改且随时间单调增加的数据，采用全新的合并策略，对数据块不进行全部合并，而是按照时间序合并成多个稍大的文件，并且设计全新的索引和过滤条件，在大幅减少了磁盘操作的同时，实现了数据的高速检索。

（3）实现了高级负载均衡策略。HBase 起初主要为互联网应用设计，通常只有一张大表，数据的分布和负载均衡策略均通过一张大表进行设计优化。而电信话单和上网记录通常需要按照计费月分表。在直接使用 HBase 时，在每个月月初时数据导入速度和查询速度非常慢。对 HBase 的 Master 均衡算法调整（根据大表数量、分区数量、数据局部性）后，性能得到极大提升。图 4.8 是在测试集群 8 台服务器时的测试结果。

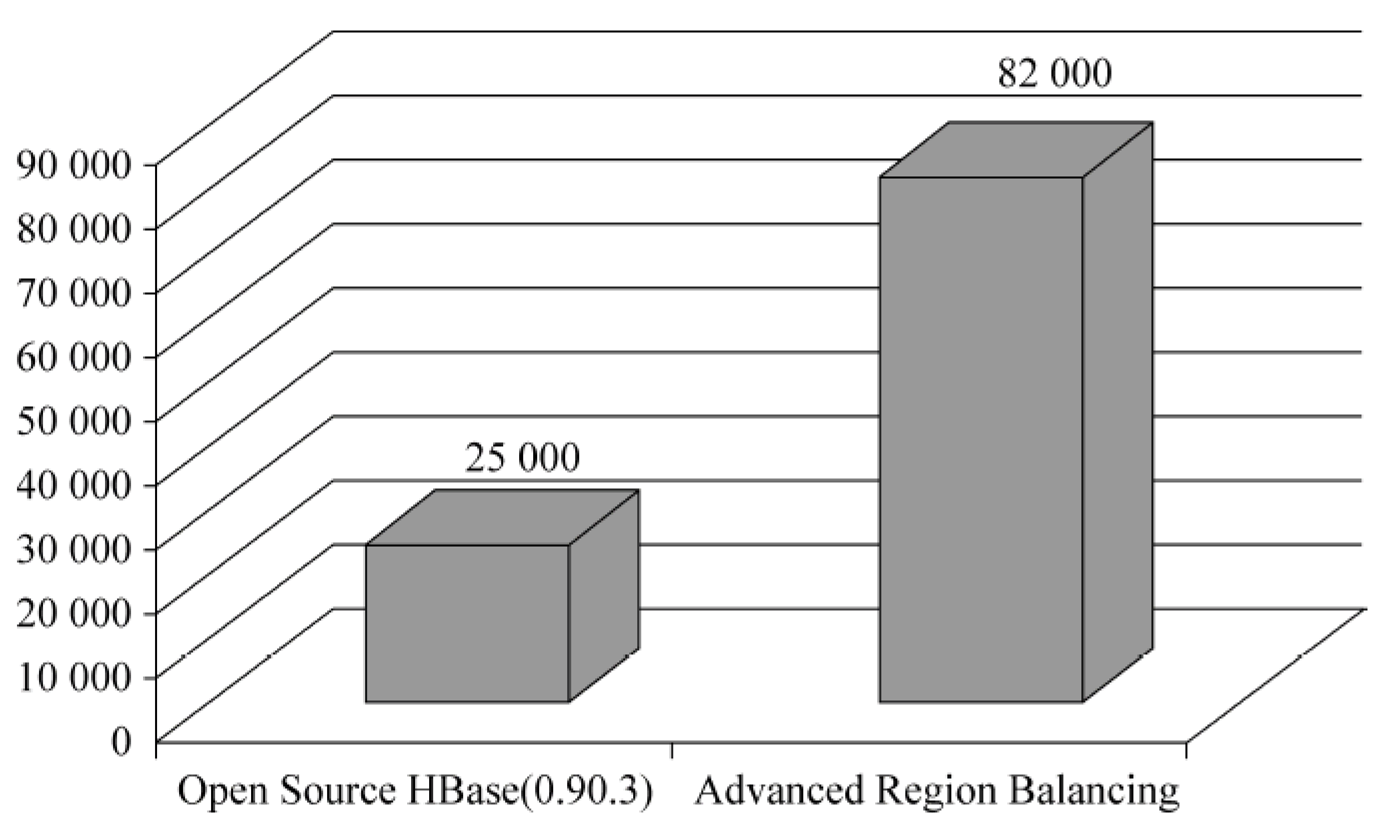


图 4.8 测试集群 8 台服务器时的测试结果

2. 高效的数据检索能力

为了支持对用户上网记录的高速统计（包括 Group-By、聚合、排序、JOIN 合并等），在 HBase 上未采用 MapReduce 而是设计了全新的 SQL 执行引擎，有效地降低了统计延时。在统计一个用户在一个月内按天统计的上网上下行流量时，返回结果在 100ms 以内。

新的 SQL 执行引擎采用分布式计算和分层聚合方法，充分利用磁盘和网络的带宽，可以最大限度发挥集群性能。

此外，为了支持按照任意字段对记录进行检索，设计实现了增量式的全文索引功能，并且支持准实时查询。上网记录导入 HBase 库中，随即就能够进行关键字和条件搜索，避免了以前的全文索引和搜索技术需要定期重建索引的问题。

3. 系统可扩展性

大规模分布式系统的最重要的要求之一是可扩展性，能够扩展到数百至数千个节点。在联通上网记录存储和查询系统中，上网记录的量在 3～6 个月内会翻番，因此可扩展性是一个硬性要求。

为了达到高可扩展性，在系统设计时，不得使用分布式事务处理等全局算法，并且任何操作需要能够容忍网络延时，特别是高并发的查询操作不得依赖多个服务器节点（不超过 3 个），否则无法进行扩展。在引入 HBase 系统时，根据上网记录的地域分布特点，对数据进行预分区，并且使用 Zookeeper（实现 paxos 算法）来维护索引信息。任何一个查询操作最多通过两个服务器。索引信息被缓存在客户端，绝大部分查询只需要访问一台服务器，因此系统的并发处理能力随着集群规模扩大而线性增加。

另一方面，为了解决集群性能受限于集群中慢速机器的问题，为 HDFS 增加了动态负载均衡能力。HDFS 在 2003 年设计之初，假设的条件是访问本地磁盘最快，而网络较慢（100Mbps）。但现代网络技术发展很快，10Gbps 的以太网开始普及，而磁盘速度没有飞速发展。因此，我们设计了全新的 HDFS 数据块选取策略，自动监控全网的点对点的访问延时，在读取数据块时，选取访问延时最低的服务器，而不总是本地磁盘。这个改进极大提高了系统的可扩展性。

4. 系统的高可用性

在实际生产环境中 HBase 存在多个稳定性问题，包括元数据（meta table）损坏、regionserver 异常退出、元数据因为竞争（race condition）导致状态不一致和 HMaster 崩溃等问题。在实际应用中，解决了所有这些稳定性问题，完善了 HBase master 处理逻辑，对不稳定的网络环境有更好的适应性，从而保证系统持续稳定地工作。表 4.1 列出了提交到 Apache HBase 社区的补丁，这些补丁针对联通上网记录集群中发现的问题，给出了相应的解决方案。

表 4.1 提交到 Apache HBase 社区的补丁

Key	Summary	Issue Type	Status	Priority
HBASE-6300	Master should not ignore event RS_ZK_REGION_OPENED when regionState is null or unexpected	Bug	Open	Major
HBASE-6299	RS starts region open while fails ack to HMaster. sendRegionOpen() causes inconsistency in HMaster's region state and a series of successive problems	Bug	Patch Available	Critical
HBASE-6289	ROOT region doesn't get re-assigned in ServerShutdownHandler if the RS is still working but only the RS's ZK node expires	Bug	Patch Available	Critical
HBASE-6049	Serializing "List" containing null elements will cause NullPointerException in HbaseObjectWritable. writeObject()	Bug	Resolved	Major
HBASE-6029	HBCK doesn't recover Balance switch if exception occurs in onlineHbck()	Bug	Resolved	Major
HBASE-5829	Inconsistency between the "regions" map and the "servers" map in AssignmentManager	Bug	Resolved	Major

目前每天记录条数在 700 亿以上，最高入库速度达到 145 万条每秒，存储采用 LZO 压缩算法，压缩比为 28%。HBase 集群每个 regionserver 有 1500 个 region。

4.3.3 优化策略

优化策略是由多种方法组成的策略组，包括多维动态负载均衡算法，增强磁盘快速顺序写方法，动态自适配数据分布和读取算法，增量式全文索引方法，分层聚合的 SQL 执行引擎和多层次、分布式统计方法等，如图 4.9 所示。

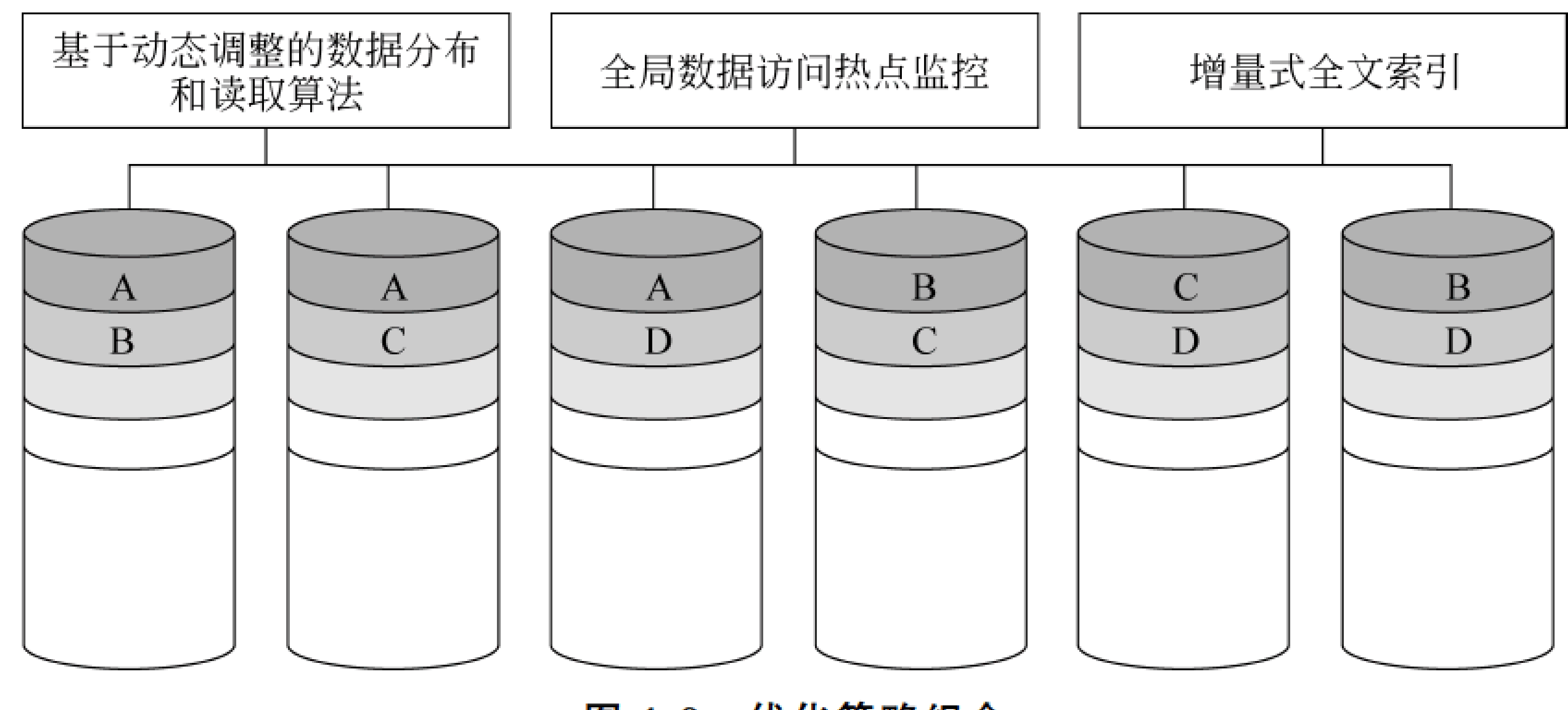


图 4.9 优化策略组合

4.3.4　HBase 与 DatabaseX 比较

我们将 HBase 开源方案与另一商用关系型数据库 DatabaseX 进行了基于二级索引查询性能比较，比较测试的环境参见附录 B。

1. 插入速度统计分析

DatabaseX 在建立局部前缀索引的情况下，插入效率与数据量大小关系如图 4.10 所示。从图中可以看出随着数据量的增大，插入效率明显降低。由图中数据推测，在边插入数据边建立索引的情况下，插入 25 亿条数据，预计需要 230 小时。

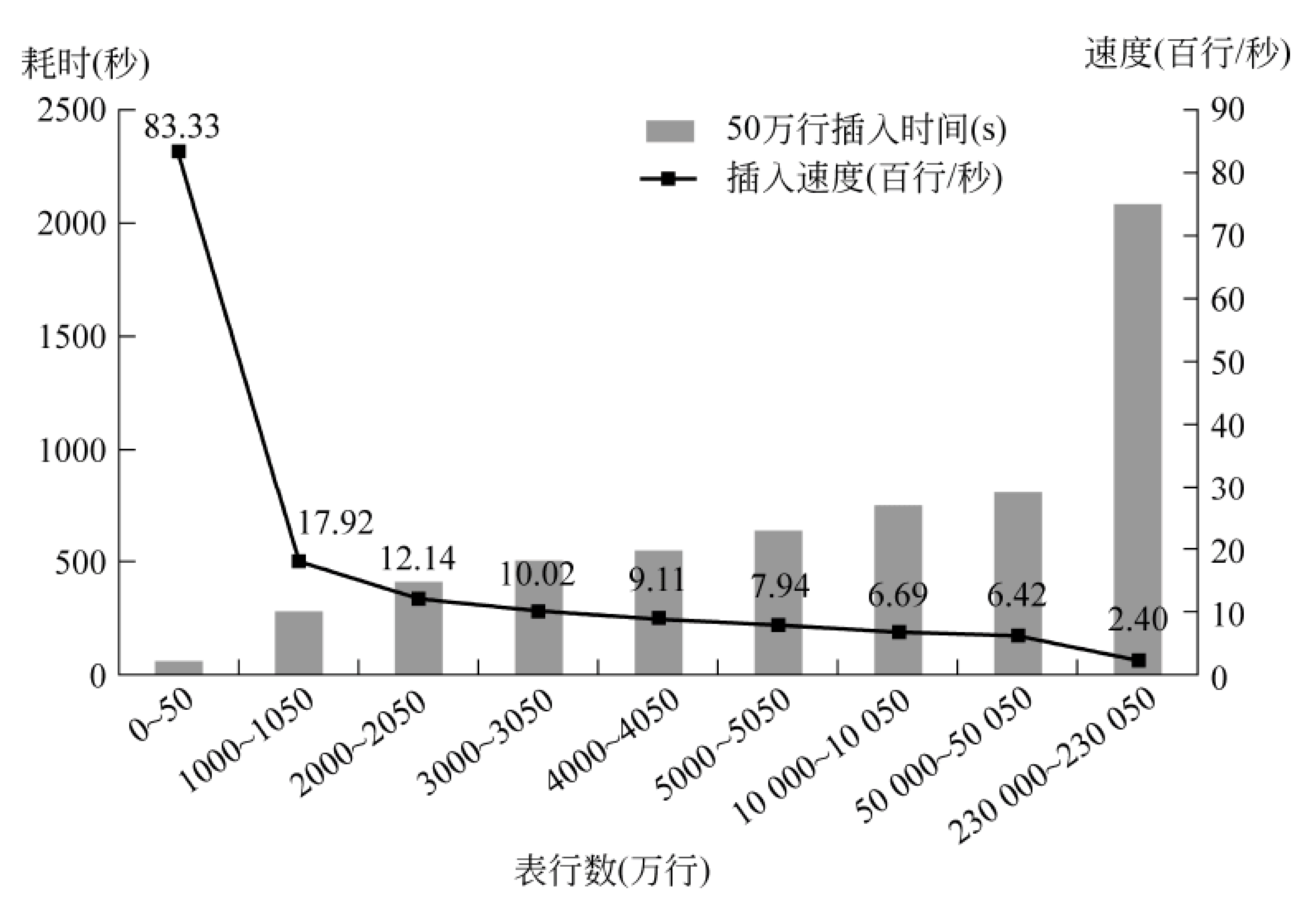


图 4.10　DatabaseX 建立局部前缀索引时的插入速度

测试中 HBase 插入 31 亿数据总共耗时 746min，平均插入速度为 69 260 条/秒。图 4.11 为插入过程中按每分钟统计平均插入速度所绘制的图，从图中可以看出，随着数据的插入，HBase 的插入速度基本是平稳的，没有明显的下降。

2. 平均响应时间

图 4.12 和图 4.13 分别是 DatabaseX 数据库与 HBase 查询在不同并发度情况下的平均响应时间，图中只选取两个数据集大小与 HBase 表大小相近的进行展示，从图中可以得出两点结论。

(1) HBase 的查询响应时间明显优于 DatabaseX 数据库，HBase 是毫秒级响应，而 DatabaseX 是秒级响应。

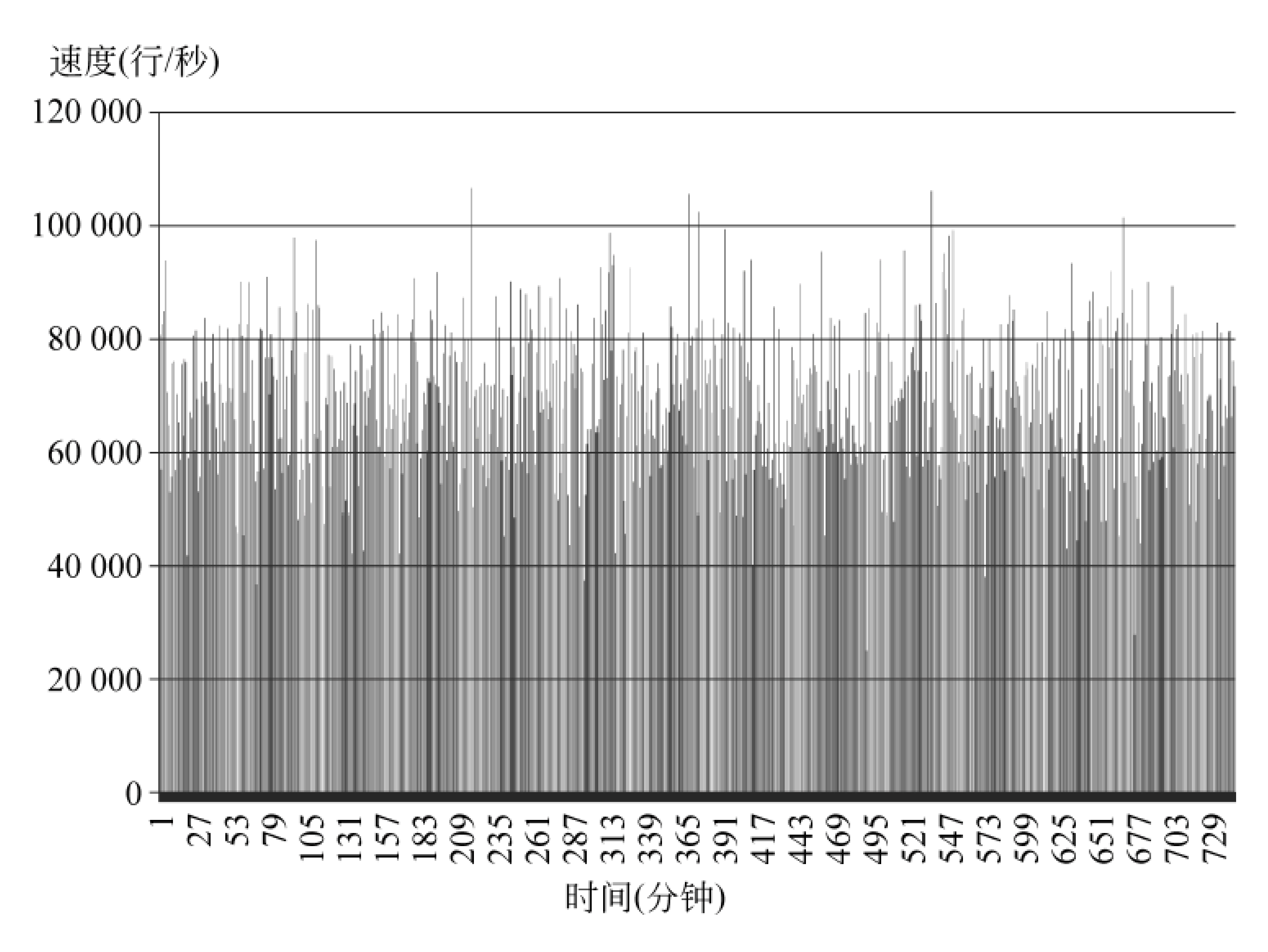


图 4.11 HBase 插入 31 亿数据每分钟平均插入速度

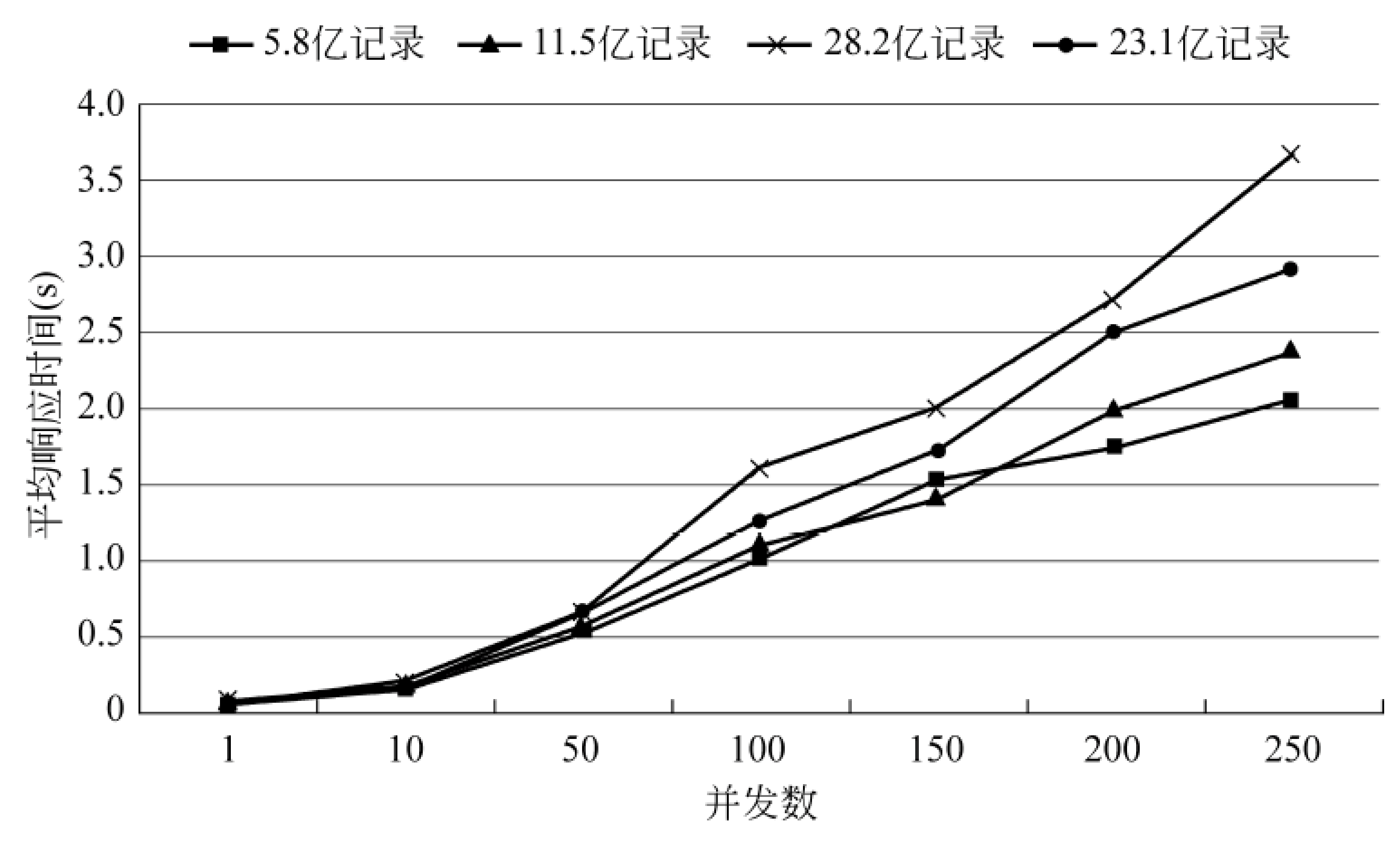


图 4.12 DatabaseX 查询平均响应时间

(2) 随着数据量的增加,DatabaseX 数据库查询在并发的情况下,响应时间会明显变长,而 HBase 的响应时间变化相对较小。

3. 吞吐量

图 4.14 和图 4.15 是在测试中 DatabaseX 和 HBase 在 250 并发下的吞吐量随着数据集大小变化的情况。从图 4.14 中可以看到 DatabaseX 数据库的吞吐量都逐渐降低,

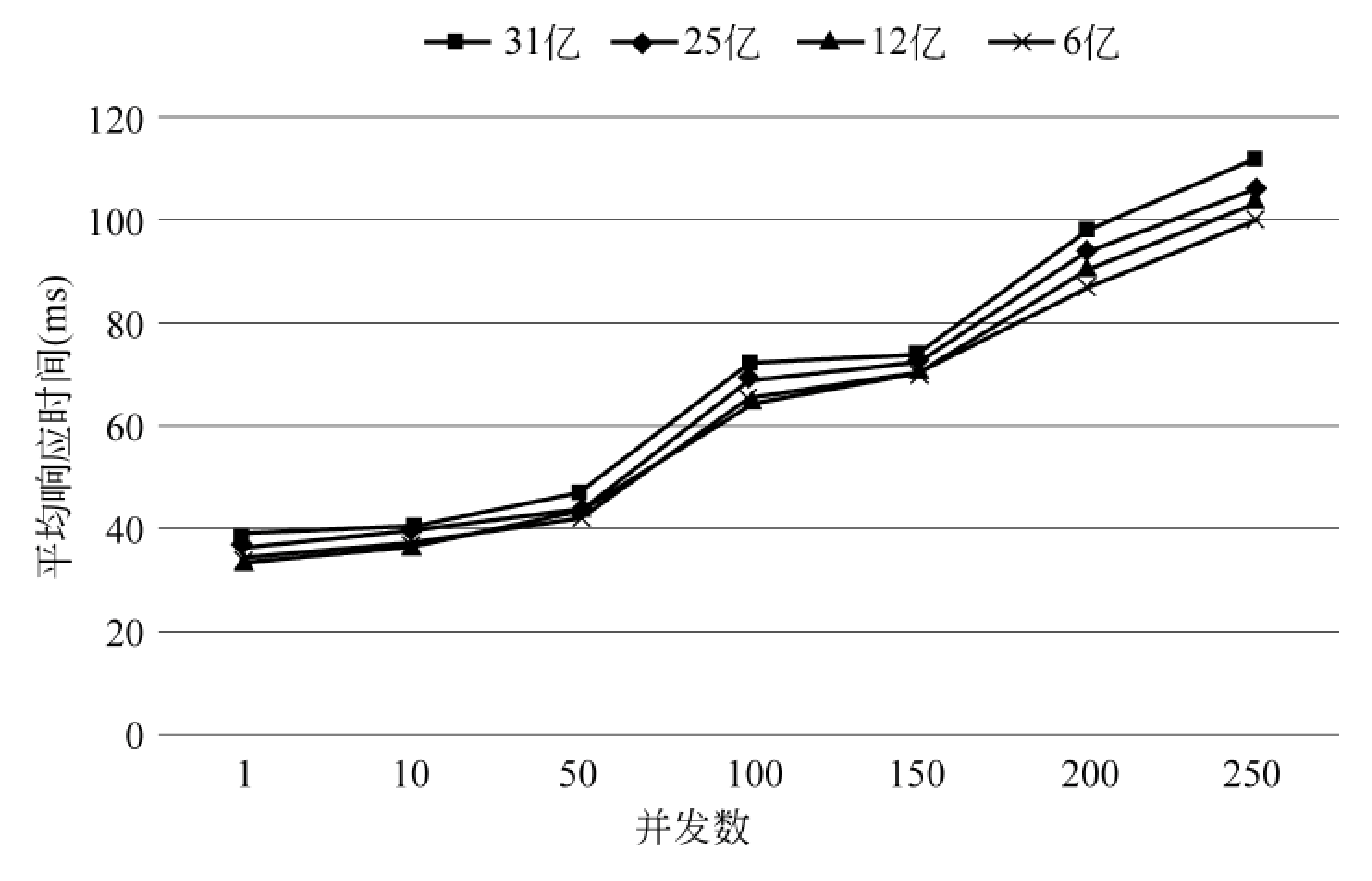


图 4.13　HBase 查询平均响应时间

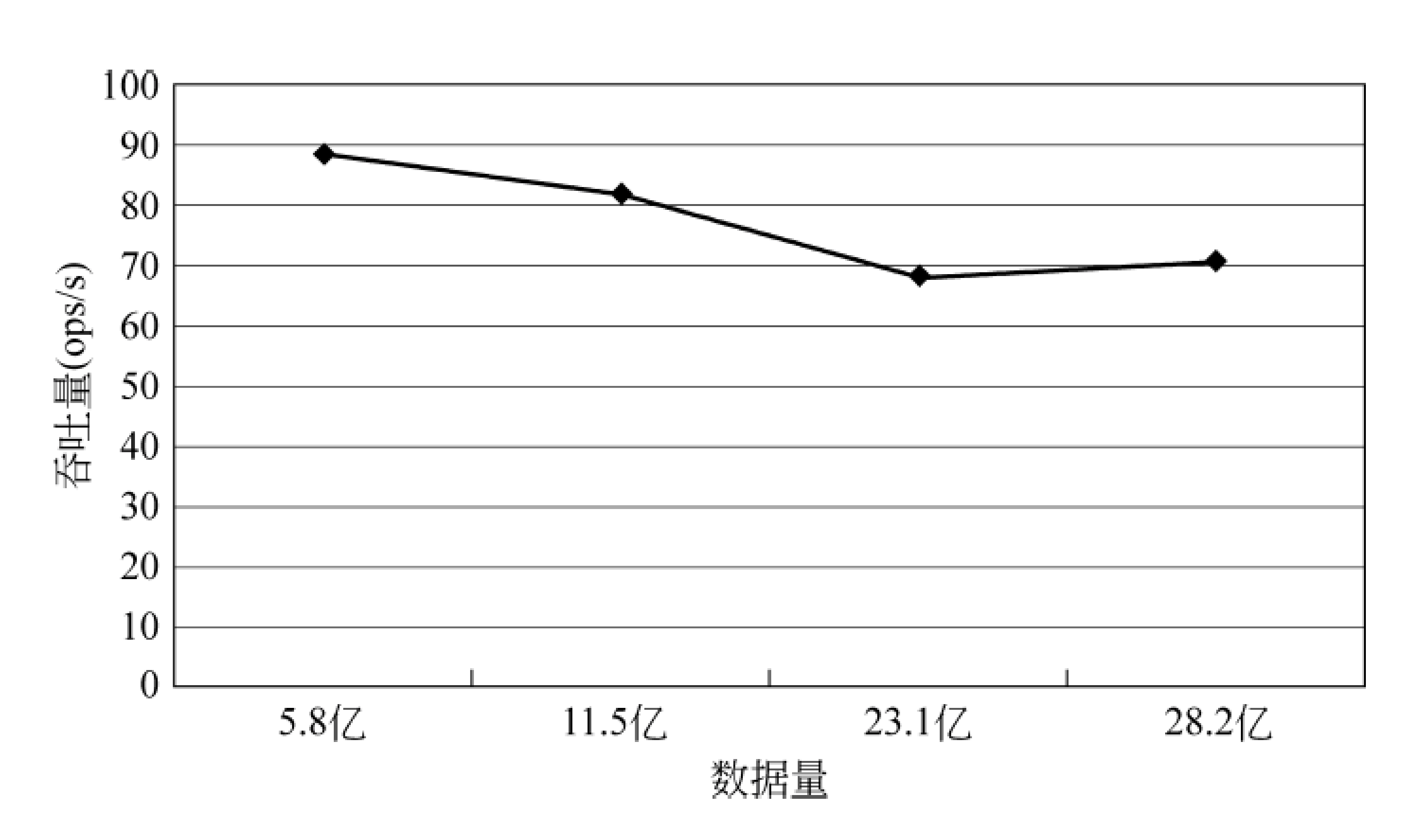


图 4.14　DatabaseX 数据库吞吐量随数据集增大的变化(并发度为 250)

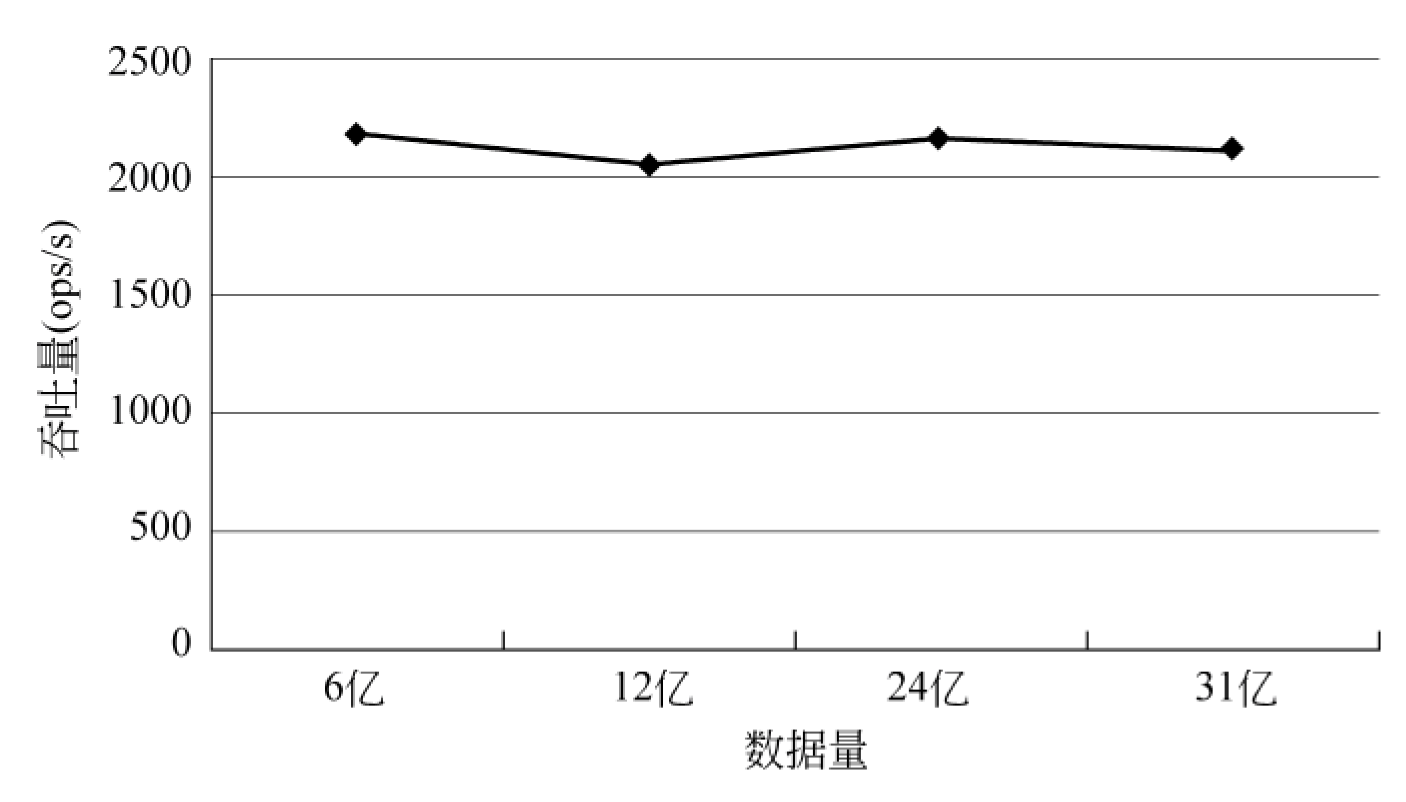


图 4.15　HBase 数据库吞吐量随数据集增大的变化(并发度为 250)

HBase 没有明显的下降趋势。从整体而言，HBase 的吞吐量明显高于 DatabaseX，这主要是由于 HBase 的响应时间比 DatabaseX 数据的响应时间低一个数量级。

4.4 流量大数据-经营与挑战

4.4.1 流量大数据经营

流量大数据经营是一项投资巨大、涉及面广、具有重要意义的活动，下面以中国联通移动互联网综合业务分析与监控平台[3]为例，该平台曾获 2013 年国家科技进步二等奖。

1. 流记录数据采集

中国联通移动互联网综合业务分析与监控平台通过在省分公司侧的 Gn 接口采集分组域用户上网数据，并进行信令和业务解析、合成，生成移动用户上网流量详单记录（Flow Detail Record，FDR）。通过进一步采集 Gi 防火墙系统日志信息，结合 FDR 记录，生成符合 IP 地址溯源要求的用户 IP 地址溯源记录。上述记录通过 IP 承载网 FTP 上传至总部平台中。总部平台对数据进行校验并入库存储。基于实时数据的存储汇总分析，总部平台在数据层之上建立以下应用服务：上网记录查询服务、IP 地址溯源服务、NET 手机号码识别、身份鉴权、个性化信息服务、统计报表与用户行为分析服务、3G 网络建设和运行分析服务。

中国联通移动互联网综合业务分析与监控平台总体架构如图 4.16 所示。

文件上传及缓存服务器将收到的 FDR 和 IP 地址溯源数据上传至数据入库子系统的指定目录中，该目录根据不同的省份进行设定。服务器同时具备缓存 7 天的能力和重传纠错功能。

FDR 包含的字段有手机号码、位置区编码、CI 号码、终端类型、流量类型、开始时间、结束时间、上行流量、下行流量、访问的网络类型、终端 IP、目的 IP、状态码、用户代理（User Agent）、APN、IMSI、SGSN IP、GGSN IP、内容类型（Content-Type）、源端口、目的端口、话单合并标识和网址/特征信息等。

采集处理设备可以识别的业务类型大类包括彩信、WAP 及 Web 网络浏览、即时消息、流媒体、邮件、网络电话、文件传送和 P2P 等。

对于 IP 地址溯源，记录中包含的字段有手机号码、NAT 开始时间、持续时间、访问时间、终端 IMEI、用户私网 IP、私网端口、NAT 后公网源 IP 地址、NAT 后源端口、NAT 后公网目的 IP 地址、NAT 后目的端口、位置区域码信息、服务区域码信息和目的 URL 等。

2. 系统部署和实施

移动用户上网记录检索与应用分析系统由中国联通研究院开发完成，在中国联通全国进行了集中部署。部署架构如图 4.17 所示。

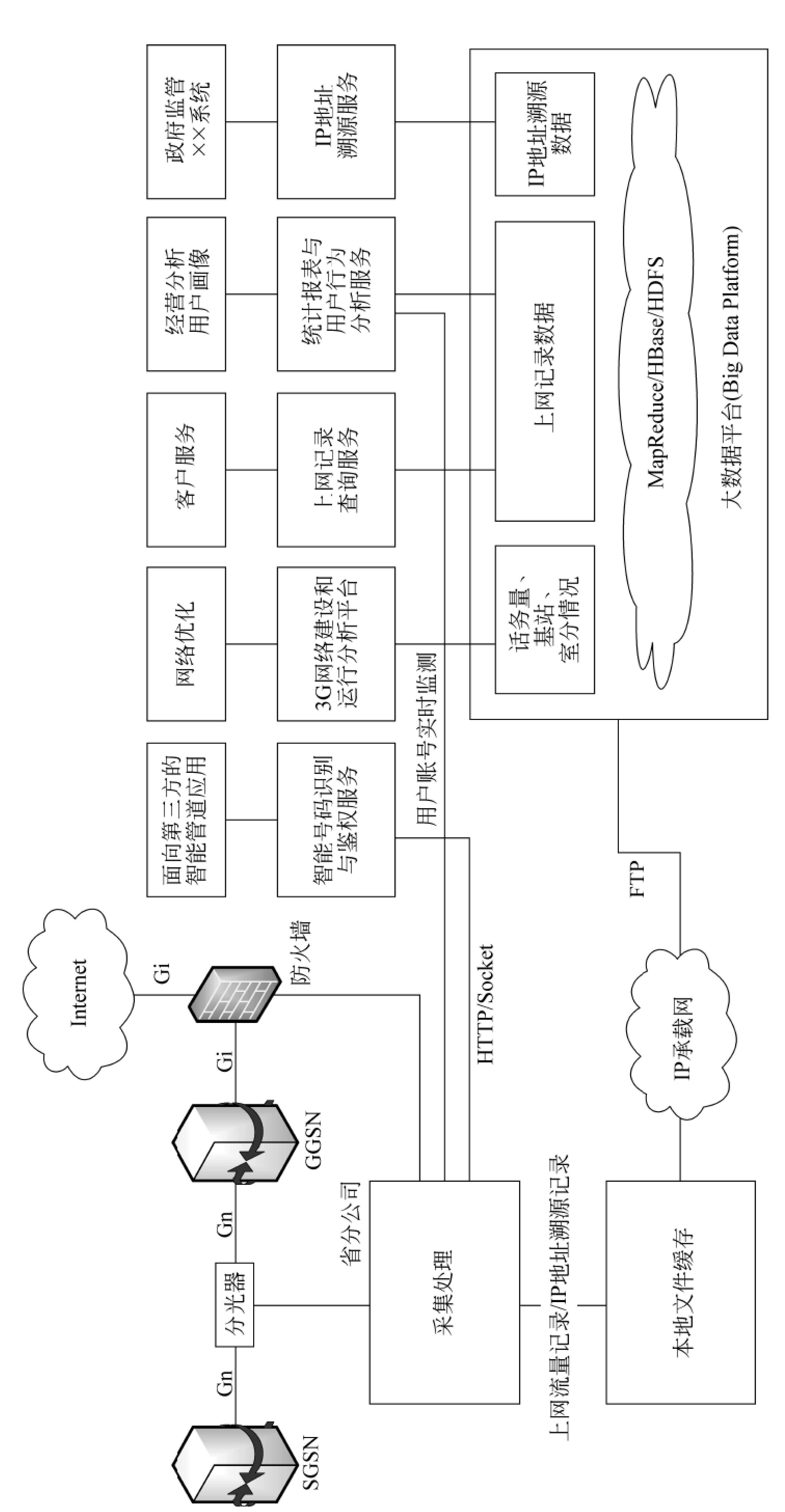


图 4.16　中国联通移动互联网综合业务分析与监控平台

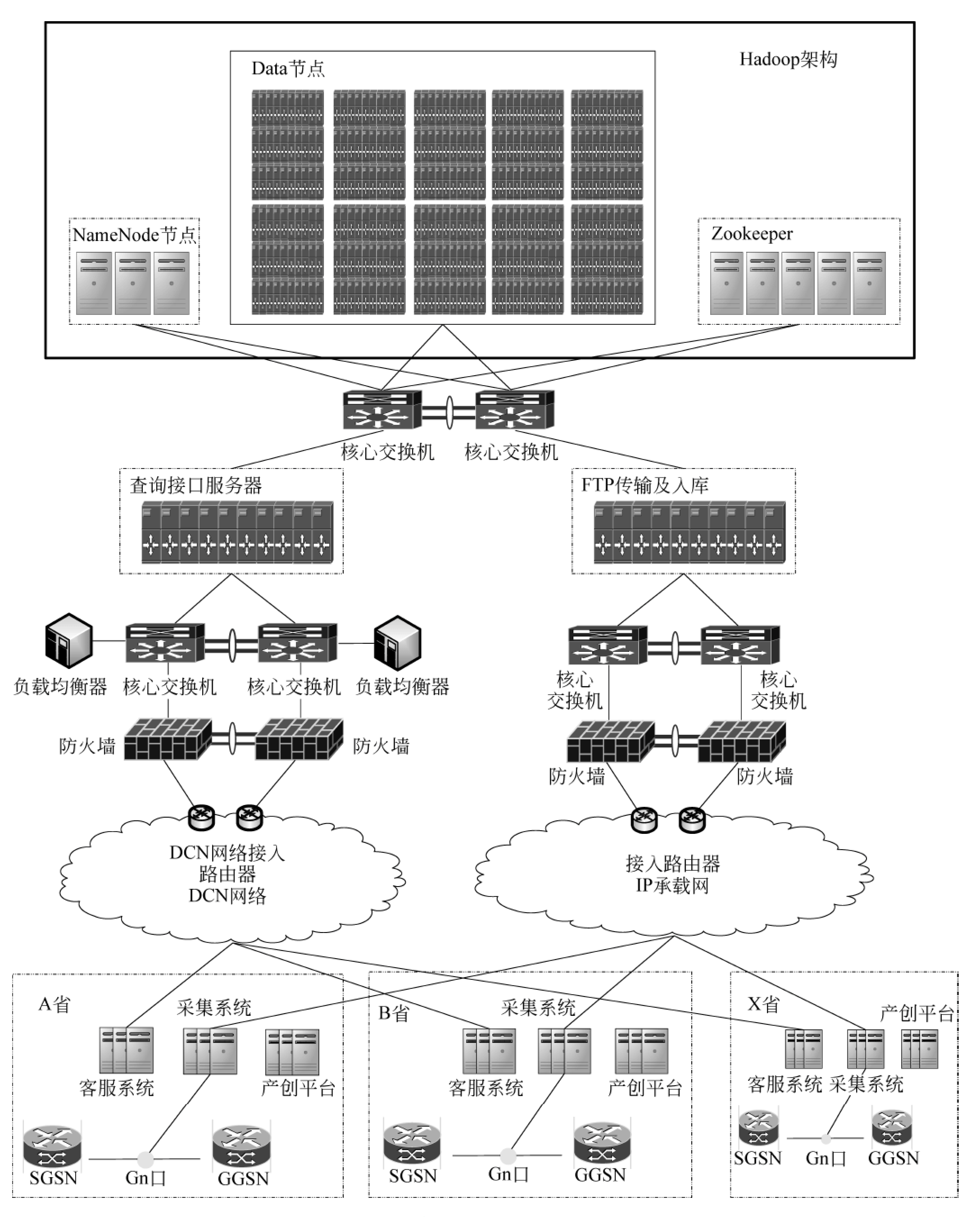


图 4.17　移动用户上网记录检索与应用分析系统

3. 系统性能指标

（1）每条用户上网记录中，用户号码填充率在 99%以上，业务类型识别率在 95%以上。

（2）CRM 计费详单与上网记录中的流量比对精度在 98%以上。

（3）上网记录查询速度一般不高于 1s（不包含用户访问查询页面的时间）。

（4）在用户访问行为发生后能查询到该条上网记录的时延小于 30min。

4. 上网记录查询与分析服务

数据查询与分析子系统主要完成上网记录查询、统计分析和 IP 地址溯源查询等功能，如图 4.18 所示。

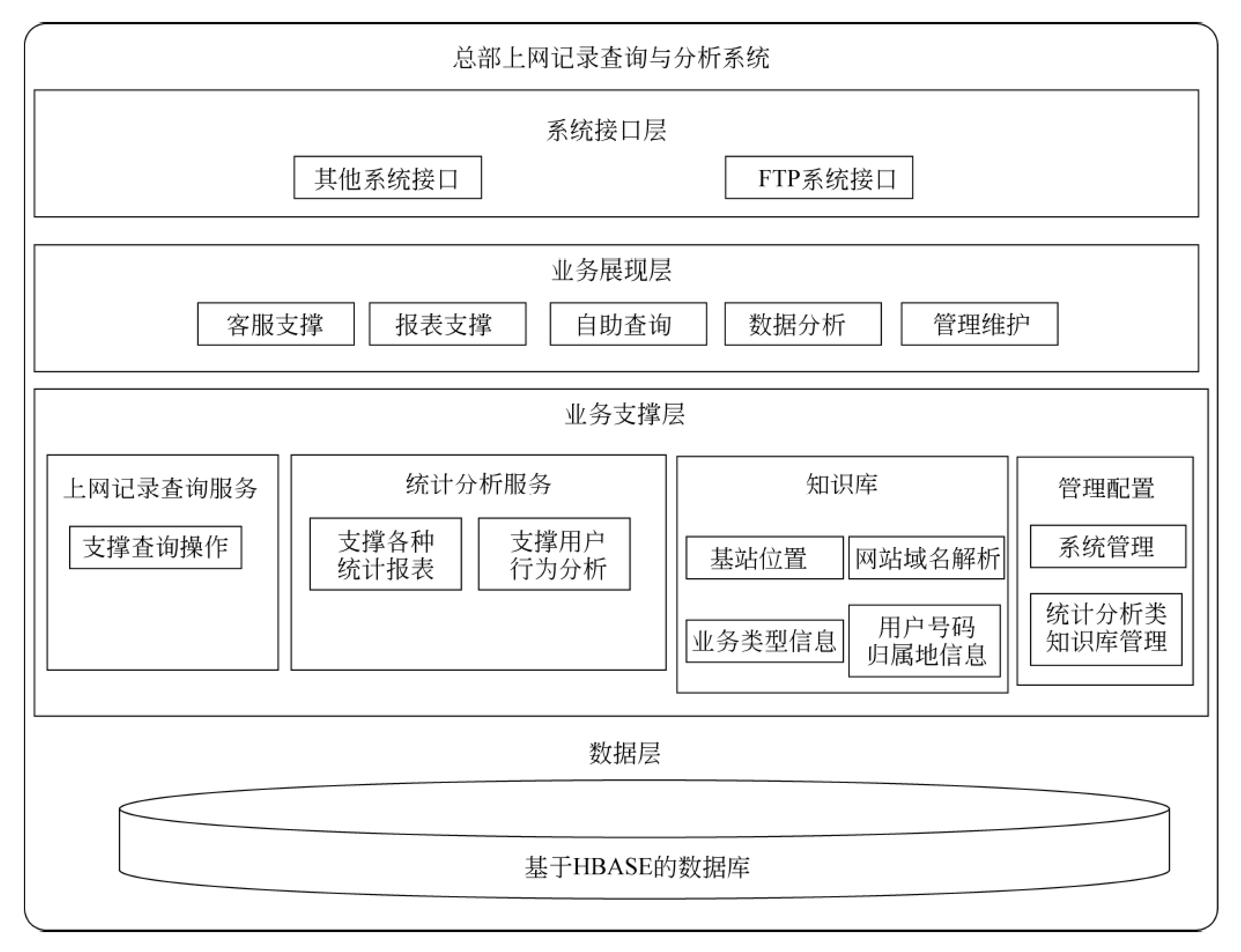


图 4.18　上网记录查询与分析系统功能

（1）记录查询。

根据其作用主要分为两大部分，一部分给最终客户提供，实现自助查询；另一部分给营业员提供，主要解决流量投诉问题。用户可以进行多种方式的查询：按用户号码、上网

方式、时间段、网络承载类型、业务类型、信息类型等查询，以及按以上各种条件进行组合查询。查询结果返回内容包括用户上网的接入地点、上网方式、业务类型、信息类型、发送流量、接收流量、总流量、开始时间、结束时间、总时长和访问网站地址等。

（2）统计分析。

通过 MapReduce 和 HIVE 技术，实现对数据流量进行统计、分析，获取统计数据，包括流量分布地区、热点业务、热门网站等具有高价值的统计分析信息，以及实现用户行为分析和描述用户画像等。

4.4.2 流量大数据—挑战

1. 大数据安全体系建设

大数据安全体系建设是保证用户隐私数据不泄露的必备条件，一般而言，系统安全机制由认证（Authentication）和授权（Authorization）两大部分构成。认证就是简单地对一个实体的身份进行鉴别判断，而授权则是向实体授予对数据资源和信息访问权限的决策过程。

随着联通大数据平台的 Hadoop 集群被越来越多地应用在生产环境的数据分析工作中，用户越来越多，使用者的身份也越来越复杂，需要考虑权限控制与数据保护等安全问题。

Hadoop 集群的安全问题主要体现在如下方面。

（1）缺乏用户和服务器之间的安全认证机制。

（2）安全授权机制。

（3）缺乏数据传输与存储加密机制。

Hadoop 2.0 增强了安全机制，Hadoop 2.0 中的认证机制采用 Kerberos 和 Token 两种方案，而授权则是通过访问控制 ACL 列表（Access Control List）实现的。

Hadoop 2.0 认证机制中，Client 与 NameNode 首次通信以及 Client 与 Resource-Manager之间首次通信时均采用 Kerberos 进行身份认证，之后切换到 Delegation Token 以降低开销，而 DataNode 与 NameNode 和 NodeManager 与 ResourceManager 之间的认证始终采用 Kerberos 机制。

Hadoop 2.0 授权机制中，Hadoop YARN 的授权机制是通过访问控制列表（ACL）实现的，按照授权实体，可分为队列访问控制列表、应用程序访问控制列表和服务访问控制列表。

2. 用户隐私保护

在构建以上商业生态系统的过程中，隐私是需要重点考虑的内容。为了解决隐私问题，真实用户 ID 转换为 anonymous ID/pseudo ID 是关键基础，即用户数据匿名化。同时真实 ID 的匿名化或者伪 ID 生成过程，可以采用定时更新机制[2]。通过这种 defender 防

御机制，可以有效挫败攻击者通过累积密文获取信息量的攻击方法。确保大数据服务合作伙伴只能获得相应的部分信息，而非用户的所有信息。

用户隐私保护的其他有效手段，包括如下。

(1) 全机器自动处理。

(2) 统计数据而非个体数据。

(3) 安全加密的数据库[3]。

3. 新商业模型

ISP 累积了大量的移动上网流量与记录。ISP 商业模式(Business Mode)发生了创新变化，从话音与 SMS 业务，转为买流量送话音短信。特别是 OTT 业务发展，如 WeChat、IMessage 和 Facetime 等，进一步促进了 ISP 的模式改变。

ISP 从基础框架提供服务转向大数据经营，如何构架以大数据经营为中心的生态系统 Ecosystem。需要吸引合作方或利益相关者企业参与，主要参与的合作方如下。

(1) 大数据服务商。

(2) 大数据分析与挖掘商。

(3) 内容提供与客户端。

经营流量数据，依靠底层健壮有效的数据采集与检测系统，和有效的大数据平台的支撑。下面介绍一下中国联通的数据采集与大数据平台的生态系统。

1) 大数据生态系统分工与共荣

运营商的推送与能力开发平台，参考互联网的开放平台的功能与实现。为大数据服务商(开放接口与平台)、大数据分析与挖掘商和内容提供与客户端提供不同层级的数据视图(Data View)。在保护用户隐私和安全前提下，各司其职，实现价值链的构建与共赢。

2) 个性化推荐系统实例

个性化推荐系统[11]是用户潜在兴趣与内容提高商的内容匹配的关键，好的推荐系统能够有效挖掘用户需求，减少用户获取内容的时间，提供用户体验。大数据分析与挖掘商和内容提供商通过个性化推荐技术，如协同过滤[12]等，分析用户历史行为，最大化推荐系统和用户体验。

4.5 流量大数据—总结

构建基于流量大数据的产业生态系统是运营商的发展方向。本章提出了基于流量大数据的生态系统产业经济建设的方案，产业合作方 partner 包括基础框架服务商 ISP、运营商的大数据服务商、大数据分析与挖掘企业、内容与客户端应用商。流量大数据产业经济依赖于健壮有力的大数据平台和流量获取与分析系统。

本章介绍了中国联通的大数据服务的关键技术及基础架构。中国联通的流量用户数

据采集与大数据平台，为中国联通的业务产生更大的价值，促成了移动用户、运营商和互联网服务公司的多赢。

参 考 文 献

[1] 梁勇，陈震，石希，李军. 高速网络流量管理系统研究. 信息网络安全，2012，7：8.

[2] Yuqian Li, Yang Liu, Zhi-fang Liu, Jiwei Huang, and Zhen Chen, The Power of Refresh: a Novel Mechanism for Securing Low Entropy PII. CMC2011.

[3] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011.

[4] Zhen Chen, Lingyun Ruan, Junwei Cao, Yifan Yu, and Xin Jiang. TIFAflow: enhancing traffic archiving system with flow granularity for forensic analysis in network security. Tsinghua Science and Technology, 2013, 18(4).

[5] Zhen Chen, Xi Shi, Ling-Yun Ruan, Feng Xie and Jun Li. High Speed Traffic Archiving System for Flow Granularity Storage and Querying. ICCCN 2012 workshop on PMECT, 2012.

[6] Jun Li, Shuai Ding, Ming Xu, Fuye Han, Xin Guan, and Zhen Chen. TIFA: Enabling Real-Time Querying and Storage of Massive Stream Data. In Proceedings of International Conference on Networking and Distributed Computing (ICNDC), 2011.

[7] Zhen Chen, FuYe Han, Junwei Cao, Xin Jiang, and Shuo Chen. Cloud Computing-Based Forensic Analysis for Collaborative Network Security Management System. Tsinghua Science and Technology, 2013, 18 (1): 40～50.

[8] Fuye Han, ZhenChen, Hongfeng Xu, and Yong Liang. A Collaborative Botnets Suppression System Based on Overlay Network. International Journal of Security and Networks, 2012, 7(4): 211～219.

[9] Fuye Han, Zhen Chen, Hongfeng Xu, and Yong Liang. Garlic: A Distributed Botnets Suppression System. In Proceedings of the IEEE ICDCS, the First International Workshop on Network Forensics, Security and Privacy (NFSP), 2012.

[10] TianyangLi, Fuye Han, Shuai Ding, and Zhen Chen. LARX: Large-Scale Anti-Phishing by Retrospective Data-Exploring Based on a Cloud Computing Platform. ICCCN GridPeer workshop, 2011.

[11] 迪特马尔·詹尼士，马库斯·赞克，亚历山大·菲化宁，格哈德·弗雷德里希. 推荐系统. 蒋凡，译. 北京：人民邮电出版社，2013.

[12] FengXie, Zhen Chen, Hongfeng Xu, Xiwei Feng, and Qi Hou. TST: Threshold Based Similarity Transitivity Method in Collaborative Filtering with Cloud Computing. Tsinghua Science and Technology, 2013, 18(3).

[13] Wenliang Huang，等. Mobile Internet Big data Platform in China Unicom, Tsinghua Science and Technology. vol. 19, no. 1, 2014.

第5章

互联网流量攻击检测关键技术

互联网流量攻击检测主要通过网络攻击检测引擎来实现。在介绍攻击检测引擎原型的实现时，主要介绍分类模块和检测模块及其关键技术。对于一般的网包接收、发送模块细节将不再详细叙述，详见第3章内容。

在讨论网包内容深度检测算法时，主要讨论深度包检测算法本身，在讨论作为引擎核心算法的多模匹配算法时，主要是 Bloom Filter 算法和 HBM 算法。

5.1 网包处理流程

一个典型的网包处理应用程序大概包含有3个逻辑层面：数据层、控制层和管理层。处理程序基本上集中在利用网包数据读写层面上的组件。

5.1.1 系统结构

攻击检测引擎整体结构设计如图5.1所示。本引擎主要包括初始化工作和运行时工作两部分。扫描匹配引擎中，初始化工作由控制层面完成。

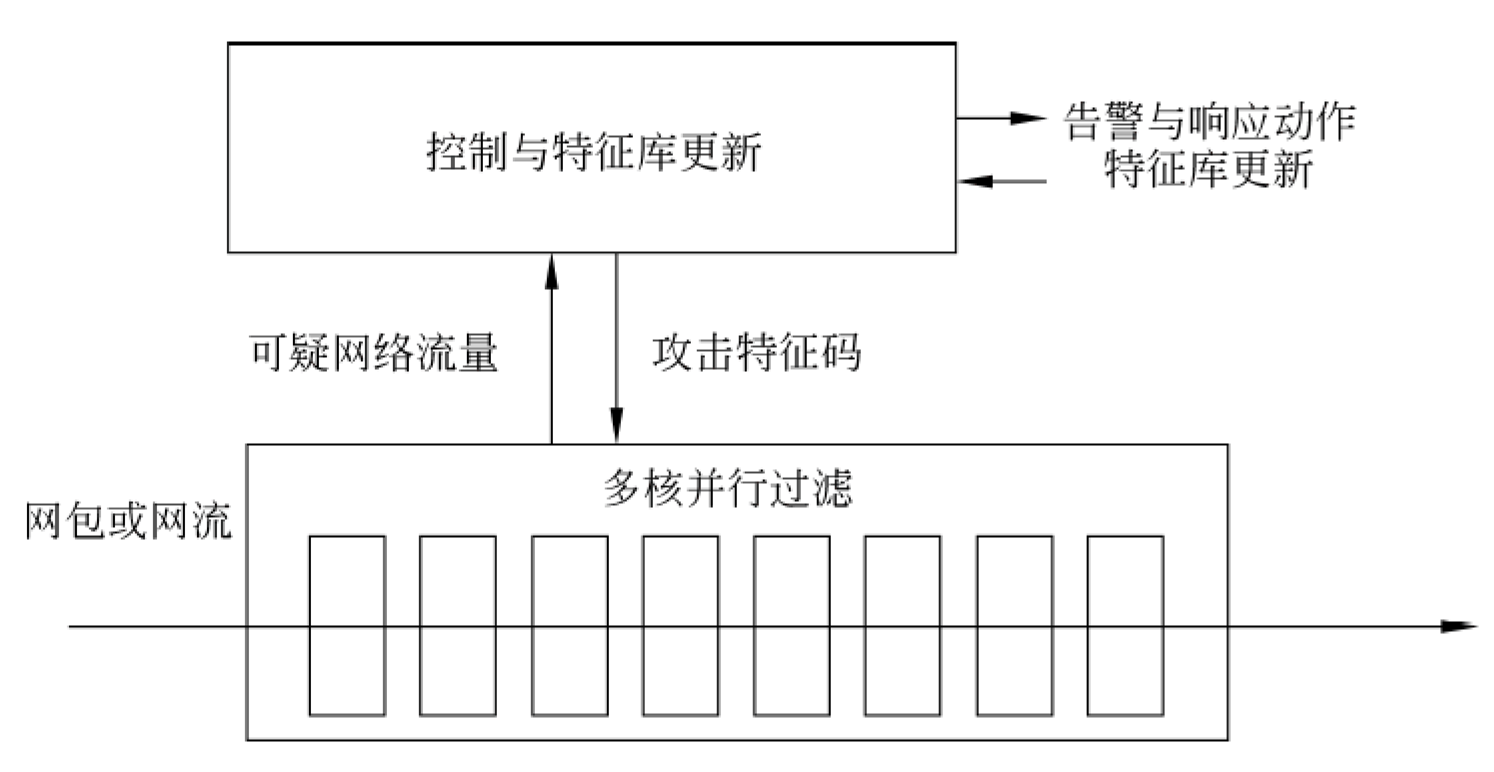


图5.1 攻击检测引擎整体结构设计

初始化工作主要完成对于攻击特征码的初始化，生成位向量（Bloom Filter 算法）或者启发式移位表（HBM 算法），同时重新将新的特征码组织成散列链表方式，减少后续精确比较时的 I/O 次数。

运行时工作主要包括网包的接收、分类、检测扫描和转发功能，其中的分类与流重组、检测扫描模块是本引擎的核心算法所在。运行时工作由数据层面的多核执行完成。

图 5.2 为多核平台的网包处理的整体流程示意图。当网包到达时，按照存储转发方式进行。每个功能模块被分配到各个核上并行执行。多个核之间依靠内存中的数据结构环通信。各个检测扫描引擎并行执行，其负载平衡可以由网包分类模块在把网包信息放入环形数据结构进行控制，也可以由各个检测扫描引擎竞争执行。

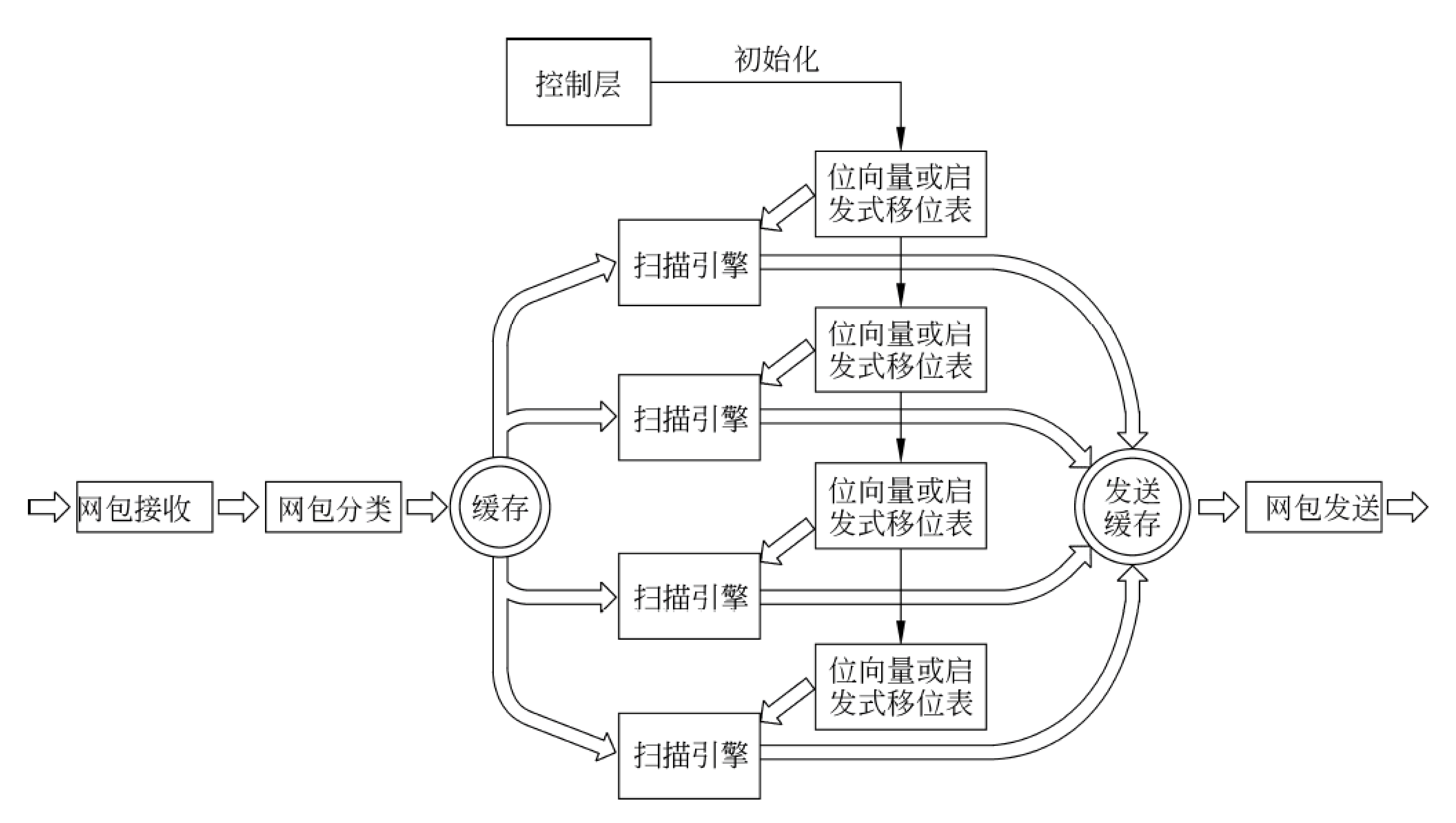


图 5.2 网包处理的整体流程示意图

1. 基于 TCP/IP 流扫描扩展功能

对于使用 TCP 连接来发起攻击来说，特征码可能会被分割到多个 TCP 网包载荷中，因此，对于这类攻击还需要进行 TCP 流状态的维护，以便即时发现跨越在 2 个或 2 个以上 TCP 网包载荷中的攻击特征码。

2. 流状态的记录

每个 TCP 流状态为 64B，512K 流共消耗 32MB 空间。32K 流状态记录空间用于散列表的碰撞分解，消耗 2MB 空间。总共消耗 34MB 空间。

表 5.1 为 TCP 流状态记录数据结构，每个流状态记录为 64B 长度，其中包括该

TCP/IP 流的五元组、五元组散列函数值、Next Flow 指针、Sequence Number、TimeStamp(用于标记陈旧无用的流状态),Remaining Length 用来指示 16B 大小的空间中存放的剩余未比较的网包载荷。

表 5.1　TCP 流状态记录数据结构

LW	Bits	Size	Field	Description
0	31:0	32	Hash Value	The hash value of five-tuple
1	31:0	32	Next Flow Pointer	Address of the next Flow state in DRAM
2	31:0	32	Source IP address	Source IP address
3	31:0	32	Destination IP address	Destination IP address
4	31:0	32	TCP Source Port	Source port
5	31:0	32	TCP Dest Port	Destination port
6	31:0	32	Protocol	Protocol (TCP, UDP etc.)
7	31:0	32	Sequence Number	TCP packet sequence number
8	31:0	32	Payload offset	Source and Destination Port
9	31:0	32	CODE BITS	Some Flags for Packet handling
10	31:0	32	Time Stamp	The time mark for the arrival packet(update)
11	31:0	32	Remaining length	Index the last remaining byte length
12	31:0	32	the first word	The first word of the last packet in flow
13	31:0	32	the second word	The second word of the last packet in flow
14	31:0	32	the third word	The third word of the last packet in flow
15	31:0	32	the fourth word	The fourth word of the last packet in flow

网流状态记录表采用定时更新,以释放相应的空间,即每隔固定时间,删除所有过时(TimeOut)的流状态记录。同时,当收到 FIN 包时,删除流状态记录,释放空间。

TCP 流状态记录存储采用散列表结构,如图 5.3 所示,以便于快速查找相应的流状态记录。这里散列函数为 CRC 单元,使用 CRC32 或 CRC-CCITT。散列表的索引是 TCP/IP 网包五元组采用散列函数后的任意 16b 值。

3. 分类器的设计和实现

分类器完成的功能包括如下。

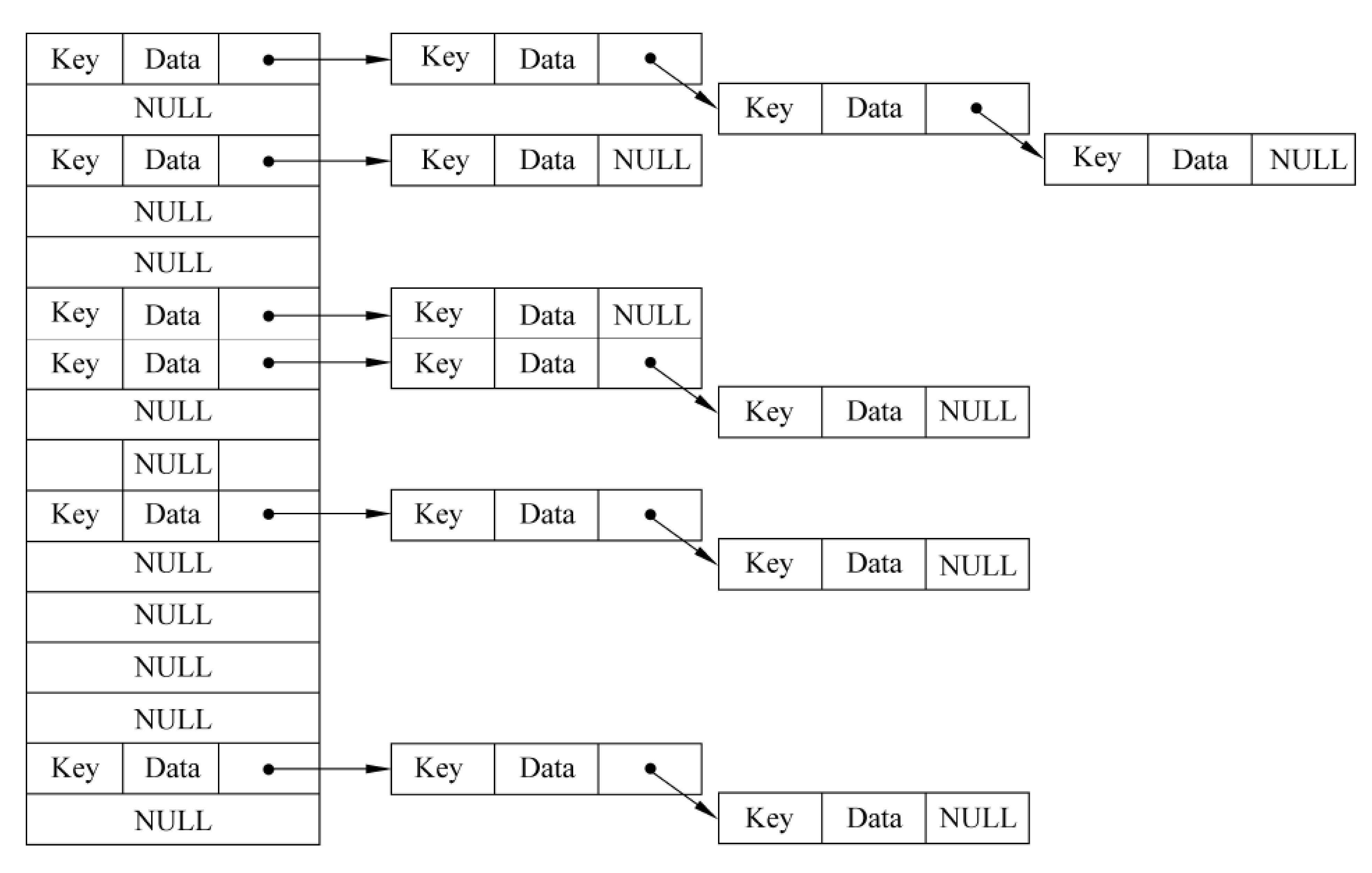


图 5.3　TCP 流状态的散列链表存储方式

（1）从接收 Scratchpad Ring 读取网包摘要信息。

（2）抽取该网包的 TCP/IP 五元组信息。

（3）对抽取的五元组进行散列操作。

（4）查找流状态信息，确定该网包是否为新到达流的网包还是已记录流的网包，然后进行相应的操作。

（5）写网包摘要信息至相应的 SRAM Ring。为了保证网包的顺序，采用线程保序的设计方式(Thread Ordered Model)，如图 5.4 所示。

4. 跨 TCP 包特征检测

根据以上两节的设计，可以做到检测跨 TCP 包特征码。原因如下所示。

（1）网络网包通过扫描线程号，由散列函数计算流状态得到，保证同一个流的网包通过同一个线程。

（2）通过信号量，将微引擎的线程模式设计成保序方式，从而保证经过该线程的同一个流的网包保序，即保证了网包载荷拼接的有效性。

（3）在 TCP 流状态记录的数据结构中，留下了空间，专用于记录同一个流上一个包的载荷尾部的数据。使得拼接过程简单。

（4）若特征码跨越 3 个或 3 个以上 TCP 包时，TCP 流状态记录能够记录前几个包载荷的拼接总和，不需要对此进行特殊的处理。

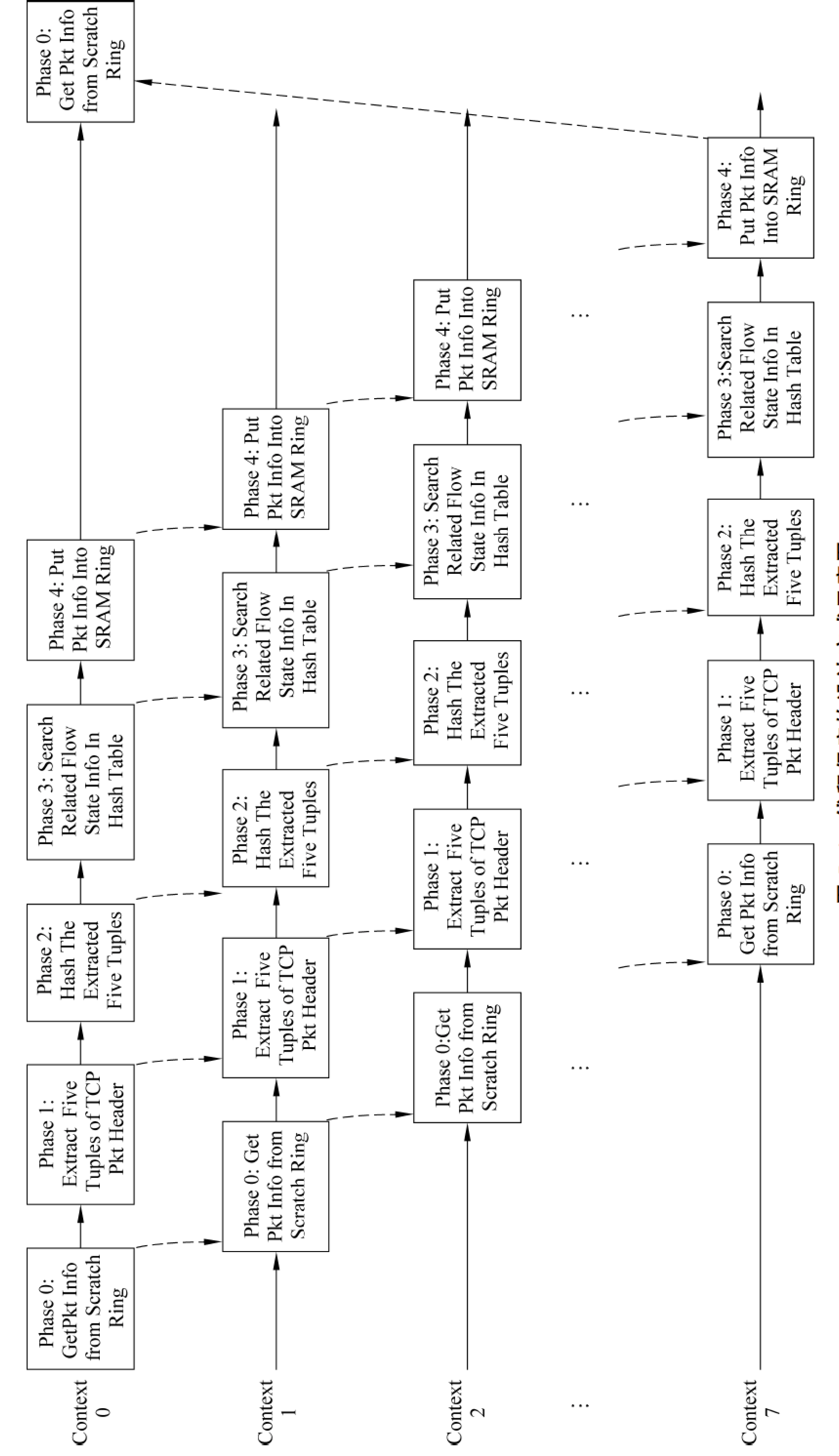


图 5.4　线程保序的设计方式示意图

5.1.2 功能模块图

匹配引擎主要功能模块及其关系如图 5.5 所示。从接收模块到达的网包先进行五元组 TCP/IP 头处理,并由此建立相应的流状态记录。接着进行网包载荷的内容匹配过程,从特征库里对网包载荷进行查找,如果特征匹配成功,则进一步对该网包进行规则处理。

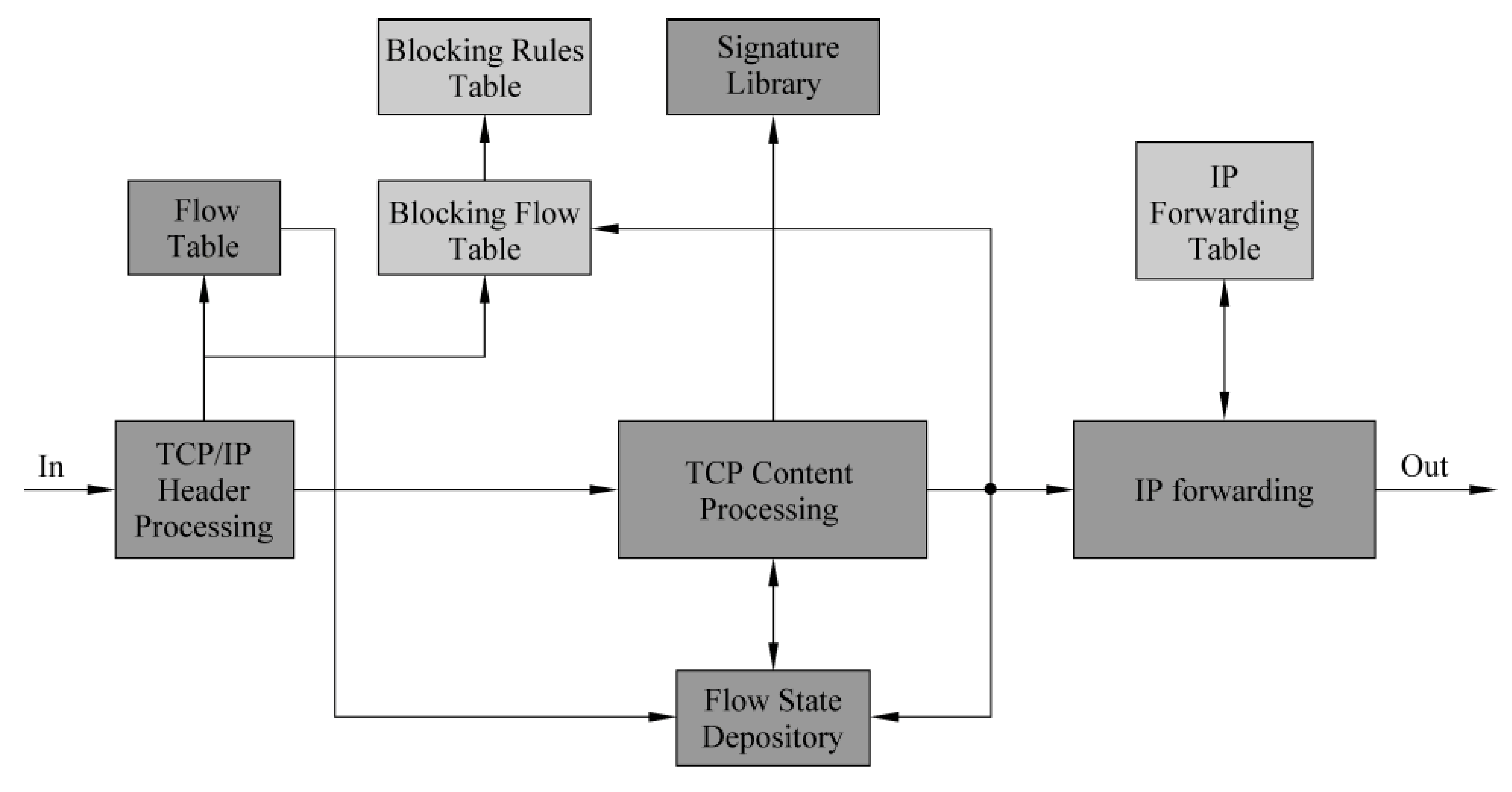


图 5.5 匹配引擎的功能模块图

5.1.3 多核组织模式

匹配引擎中多核组织模式采用并行模式,如图 5.6 所示。并行模式有很高的并行度和很高的吞吐量,但是付出的代价是数据包的乱序。

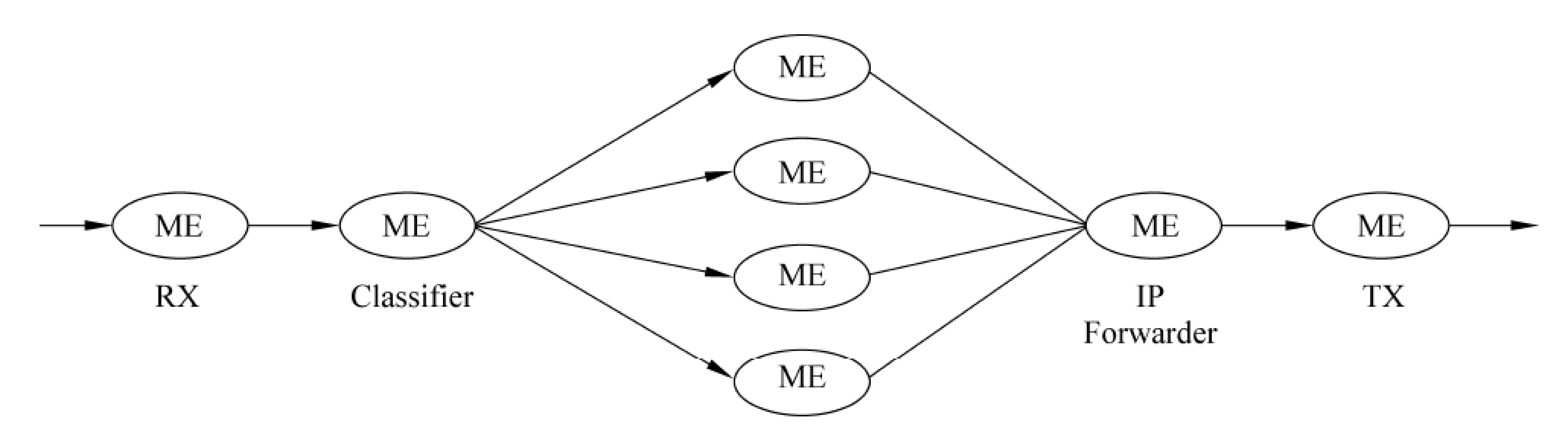


图 5.6 匹配引擎中的多核分工组合(逻辑)

深度包检测模块并行执行,并且还可以根据空闲处理核数量和速度需要调整其配置数量。同时,这样的组织方式,可以方便插拔式深度包检测模块加入到已有的系统中去。

5.2　多模匹配算法概述

深度包检测的核心算法是多模匹配算法，即检测网络网包载荷中是否含有某一个特征码。

匹配多特征字符串的算法称为多模匹配算法（Multi-pattern Matching Algorithm）[20],[24]，即对于输入的字符串，扫描其是否包含预先给定的若干特征字符串之一。这里的字符指的是广义上编码单位所能表示的所有二进制代码。在有些文献中，也将蠕虫携带的特征字符串代码称为特征码（signature）[19]。

当前主要的多模匹配算法有三条思路。分别是 Bloom Filter 算法及其改进；AC 算法及其改进；利用失效字符和启发式移位思路的算法，也就是 BM 算法在多模情况下的推广形式。下面对于这三类算法分别加以介绍。

5.2.1　Bloom Filter 算法及其改进

Bloom Filter 算法[9]是由 B. Bloom 在 1970 年提出的，主要采用多散列计算降低误判概率的思想。算法在初始化时对每个特征字符串均做多次不同的散列运算，在位矢量（bit-vector）中将这些散列函数值所对应的位进行标记，在进行匹配时，对输入字符串做和初始化时相同的多次散列运算（可并行执行散列运算，以提高速度），只要有一个散列值未在位矢量中出现时，则此次比较失败；若所有散列值均在位矢量中标记时，则将输入字符串和所有特征字符串进行字符对字符的精确比较，以确定是否匹配。

Bloom Filter 作为一种简单有效的并行多模匹配算法，被广泛用于各种硬件定制平台中。针对各个不同的应用环境有不少的改进和变种，如计数 Bloom Filter 算法和多级 Bloom Filter 算法。前者将位矢量的每一位变成计数器，用以动态增减特征码的个数；后者将多个 Bloom Filter 过滤器组成级联的方式，用以在尽可能不增加时间开销的前提下，减少误判概率。

5.2.2　AC 算法及其改进

AC 算法[10]是由 A. Aho 和 M. Corasick 在 1975 年提出，主要采用了状态机（DFA）的思想。算法根据特征字符串集合生产状态机，根据每次输入的字符转移状态，直到找到成功匹配。记 n 为输入字符串的长度，该算法的时间复杂度为 $O(n)$，不随特征字符串的多少和长度而改变。

AC 算法能漂亮地给出理论上的最坏情况的时间复杂度，但是其算法中状态机的空间要求往往很大，而且实际情况下平均性能往往也不如下面介绍扩展 BM 算法性能，对于 AC 算法的改进主要在于有效地压缩状态机空间，减少空间复杂度。

5.2.3 BM 算法的推广型

BM 算法的推广型沿用了经典的单模匹配 BM 算法的想法。其主要思路是检测失效字符和启发式移位。单字符串匹配的 BM 算法[11]是由 R. Boyer 和 J. Moore 在 1977 年提出的。算法思想是在每次匹配中,从右往左反向逐个比较字符,当发生匹配失败时,根据此字符的信息以及之前比较过的字符串后缀(即字符串最右部分)信息,进行最大限度的向后移动比较指针,从而减少无用的比较次数。

BM 算法在多模匹配中的推广形式比较著名的有 WM 算法[12]、Setwise BMH 算法[13]和 AC-BM 算法[14]。这 3 种算法的不同之处在于如何组织启发性移位表(Heuristic Skip Table)。

(1) WM 算法将每个特征字符串的后缀字符(最后 2 或 3 个字符)组成字符块,来建立移位表。

(2) Setwise BMH 算法则是将后缀字符通过 Trie 树的数据结构建立移位表。

(3) AC-BM 算法则是通过状态机建立移位表。

当前,很多匹配引擎都是采用 BM 的简单推广算法来实现。例如,RSI 算法[15]、MDH 算法[16]和 FNP 算法[17],可以看作是对 WM 算法的一种继续改进。

本章提出的 HBM 算法,也是 BM 算法在多模匹配下的一种推广,使用 2 张启发式移位表,在匹配失败时,长距离移动比较指针,减少比较次数。和同为 BM 算法推广型的其他算法(例如 WM 算法等)相比,一方面利用散列方式组织启发式移位表,减小空间开销;另一方面在移位表的构造和使用上更加符合 BM 算法的思想,减少比较次数,提高算法效率。因此,HBM 算法可以在空间开销大大降低的情况下,得到更优的时间性能。

5.3 基于 Bloom Filter 的匹配引擎

5.3.1 Bloom Filter 算法

Bloom Filter[9][19]在理论上是一种高速的分类查找算法,它的并行性极好,并且支持巨量的匹配特征数目,同时结构较为简单,有利于具体实现。

Bloom Filter 是一种容错的散列查找,它特别适用于大多数匹配特征都被拒绝的情况。假设有 n 条匹配特征,为此建立一个 Bloom Filter 检测模块,BF 中包含一个 m 位的位向量(Bit Vector,BV),对每一条匹配特征,用 k 个散列函数对它进行运算。如果这些散列函数的输出是$[1,m]$之间的值 $t_1,t_2,\cdots,t_k$,把 BV 中对应的第 $t_i(i=1,2,\cdots,k)$位置 1,k 个散列函数最多把 BV 中的 k 位置 1,n 条匹配规则最多把 BV 的 nk 个位置 1,要注意的是:一个 BF 只有一个 BV。过滤的时候把输入位序列也用这 k 个散列函数进行运算,

得到 $t_1',t_2',\cdots,t_k'$，如果 BV 中对应的第 $t_i'(i=1,2,\cdots,k)$位全是 1，那么这个位序列可能与要查找的特征是匹配的（也就是存在误判 False Positive）；如果 BV 中对应的 $t_i'(i=1,2,\cdots,k)$位不全是 1，那么这个位序列肯定和要查找的特征是不匹配的（也就是不存在漏判 False Negative）。由于大多数位序列都是被拒绝的，所以误判概率很小，平均查找次数（时间）不会受多少影响，而空间开销却大大减少了。可以证明，当 $n\cdot k=m\cdot \ln 2$ 时，Bloom 过滤器效率达到最高，误判的概率为 $f=(1/2)^k$。

鉴于特征码的多样性和多变种性，Bloom 过滤器必须实时动态地修改其匹配特征数据库，动态更新病毒特征序列，为此需要采用计数 Bloom 过滤器技术（Countable Bloom Filter）。具体修改如下：在 Bloom 过滤器中除 m 位的位向量之外，增加一个计数数组（Counter Array，CA）记录 BV 上每一位 n 条匹配特征计算散列后映射上的次数。如果要从 n 条匹配特征中减去一条匹配特征，只要看该匹配特征散列值之后在某一位上是否为 1，如果是，则 CA 在该位上的计数值减 1，当 CA 该位计数值减为 0 时，则将 BV 上该位置 0。如果要从 n 条匹配特征中增加一条匹配特征，则将对于 BV 的操作同上，同时更新 CA 记录的每一位上的次数。

1. 并行 Bloom Filter 算法

定义 1（Bloom Filter）：一个 Bloom Filter 用来表示一个域 U 上 n 个元素的集合 $S=\{S_1,S_2,\cdots,S_n\}$。其中包括一个 m 位的数组（Bit Vector），该数组初始化全设置为 0。Bloom Filter 使用 k 个独立随机的散列函数 $h_1,\cdots,h_k$，这些散列函数映射每一个 $s\in S$ 到范围$[0,m-1]$。对于任何 $1\leqslant i\leqslant k$，BV 中对应 $h_i(s)$位都设置为 1。Bloom Filter 可能出现一个误判（False Positive），就是指将并不存在于 S 中的元素 x 判断为属于 S，但是它并不会生成一个漏判（False Negative）事件。

考虑到散列函数是完全随机的假设，记 $p=\mathrm{e}^{-kn/m}$，那么误判可能的概率函数 f 如式（5-1）：

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k\approx(1-\mathrm{e}^{-kn/m})^k=(1-p)^k \tag{5-1}$$

2. 计数 Bloom Filter 算法

定义 2（计数 Bloom Filter）：一个计数 Bloom Filter 是一种增强型的 Bloom Filter，其中为了可测量性原因，增加了与位向量（Bit Vector，BV）每一位相对应的计数器向量。无论计数 Bloom Filter 增加或删除一个成员，都需要增加或消耗计数器来保证和位向量的位一致。只有当计数器从 1 变为 0，它才清除 BV 中的对应位。因此，这些 Bloom Filter 计数器只有在增加或者删除字符串时才改变。

5.3.2　参数选择

在给定特征码数目 n，位向量长度为 m，考虑到散列函数是完全随机的情况下，通过

改变并行散列函数个数 k，来使得误判概率函数 f 尽可能地小。记 $p=e^{-kn/m}$，将式(5-1)两边取对数得：

$$g = \ln f \approx \ln (1-e^{-kn/m})^k = k\ln(1-p) = -\frac{m}{n}\ln(p)\ln(1-p) \tag{5-2}$$

当且仅当 $p=1-p$ 时，g 取最小值，即判概率函数 f 取最小值。此时散列函数个数：

$$k = (m/n) \cdot \ln 2 \tag{5-3}$$

误判概率的最小值约为

$$f_{\min} \approx (1/2)^k = (1/2)^{(m/n)\cdot\ln 2} \tag{5-4}$$

5.3.3 散列函数选择

Bloom Filter 算法的散列函数选择，关系到系统误判概率和性能参数。散列函数的计算应该尽可能简单快速，取值应该尽可能平均，k 个散列函数间的相关性应该尽可能地小。

在匹配引擎设计中，我们尝试了 3 种散列方法。实现 8 个散列函数运算操作，采用 MD5 中的子函数方法需要 34 个微引擎时钟周期，而使用 CRC 单元需要 152 个微引擎时钟周期，同时 CRC 单元有限，无法由多个微引擎并行使用。而 Ramakrishna-Fu-Bahcekapili 函数在适合字节操作的 NP 上实现起来不是很方便，该方法更加适合硬件并行逻辑操作的平台。因此，引擎采用第一种方法，获得较高的计算速度和散列分布效果。

3 种散列函数的详细计算方法如下。

1. MD5 的子函数 *F*、*G*、*H* 和 *I*

用 & 表示与(AND)操作，～表示非(INVERSE)操作，| 表示或(OR)操作，∧ 表示异或(XOR，EXCLUSIVE OR)操作，在 Rivest's MD5(RFC1321)[37] 中的 F、G、H 和 I 函数如下：

$F(X,Y,Z)=(X\&Y)|(\sim X)\&Z)$；

$G(X,Y,Z)=(X\&Z)|(Y\&(\sim Z))$；

$H(X,Y,Z)=X$^Y^Z；

$I(X,Y,Z)=Y$^$(X|(\sim Z))$。

在实现中，可以将输入字符串切成若干段，替换以上 4 个函数中的 X、Y、Z，从而产生 4 个、8 个乃至更多的散列函数。

2. Ramakrishna-Fu-Bahcekapili 散列函数

设位字符串 $X=(x_1,x_2,x_3,\cdots,x_b)_2$，用“.”表示 AND 操作，^ 表示 XOR 操作，其散列函数定义如下：

$h_r(X)=r_1\cdot x_1$^$r_2\cdot x_2$^$\cdots r_b\cdot x_b$，其中$(r_1,r_2,\cdots,r_b)$是$[0,m-1]$之间随机生成数。

这样，每个散列函数包含有 b 个不同的随机数。

3. CRC 单元和散列单元

多核处理器上 CRC 单元在执行数据通道上并行执行。它包含有两个操作数，执行一个 CRC 操作，再写回结果。在 CRC 单元中，可选用两个特征多项式，即 CRC-CCITT 和 CRC-32。通过初始化设置 CRC_Remainder Local CSR（控制和状态寄存器），不同的散列函数能够通过使用同一个特征多项式来构造。CRC 操作的 32 位结果数据回写到 CRC_Remainder Local CSR。

散列单元的散列函数可以输入 48 位、64 位或者 128 位数据，分别生成一个 48 位、64 位或者 128 位散列索引。发送请求命令给散列单元把传输寄存器的数据输入散列单元中，然后基于数据进行散列操作，并将结果写回到传输寄存器。

5.4　基于 HBM 算法的匹配引擎

5.4.1　记号和假设

记特征字符串数为 n，长度为 L，块大小为 b；记特征码为 $S_0, S_2, \cdots, S_{n-1}$，其中第 k 的特征码 $S_k = s_{k0}s_{k1}\cdots s_{k(l-1)}$；记 Delta1 表长度为 m，第 i 项记为 $d_1(i)$；Delta2 表长度固定为 $L-b+1$，记第 t 项为 $D_2(t)$。取散列函数 $y=h(a_0a_2\cdots a_{b-1})$，为一个字符块（$b$ 字节）到数值$[0, m-1]$的一个函数。假设所有标号的下标均由 0 开始。

为方便下文叙述，现做以下定义。

定义字符块比较的“成功”和“失败”：在多模匹配情况下，对于每次字符块的“比较”不再是字符与字符的一一比较，而是寻找输入字符块在 Delta1 表中的值是否满足一定的条件，如果满足，称此次比较为一次“成功比较”；如果不满足，称此次比较为一次“失败比较”。

定义对子字符串的一轮匹配的“成功”和“失败”：对于输入字符串，取出和特征字符串相同长度的子字符串，并从右向左对每个字符块进行比较，称为一轮“匹配”。如果每次字符块的比较均成功，则称此轮匹配“成功”，还需要和特征字符串进行精确比较；如果某次字符块比较失败，则称此轮匹配“失败”，可以向右移动子字符串的起始位置，开始下一轮的匹配。

5.4.2　BM 算法回顾

虽然 BM 算法是一种经典的单模匹配算法，并不适合直接应用到多模匹配中，但是由于本章提出的 HBM 算法的主要思想来自于 BM 算法，在描述 HBM 算法前，先对 BM 算法加以简单介绍，以便于 HBM 算法的理解。

BM 算法是一种经典的高速字符串匹配算法。它和 KMP 算法[38]类似，都是在比较

过程中,一旦发现某个字符不同,就将比较的位置尽可能地向后移动,通过减少比较次数来降低算法的时间开销。所不同的是,BM 算法在每一次匹配字符串时,总是从后往前开始比较。

在 BM 算法中,使用 2 张表来存放指针移动的位数。当某次字符比较不同时,Delta1 给出的是根据所比较字符的信息,至少可以向后移动比较指针的位数;同时,Delta2 表给出的是根据当前子特征字符串信息,至少可以向后移动比较指针的位数。

在 Delta1 表中,每个字符有且仅有对应的一项(即 Delta1 表含有所有可能出现的字符的值的项数),其值为特征字符串最后出现该字符的位置。即记特征字符串 $pattern=a_0a_1a_2\cdots a_{L-1}$,当存在 k,满足 $a_k=X$ 并且 $a_i\neq X(i=k+1,k+2,\cdots,L-1)$时,字符 X 在 Delta1 表中对应的移位值 $D_1(X)=L-1-k$;否则 $D_1(X)=L$。

从定义可以直观看出,一旦发现当前待比较字符 X 不同于特征字符串相应位置的字符,就可以将当前指针向后移动 $D_1(X)$位。

Delta2 表的长度为特征字符串的长度。表中的项记录在当前位置上字符比较发现不同时,可以向后移动比较指针的位数。

为了便于描述,将字符串 $pattern=a_0a_1a_2\cdots a_{L-1}$,扩充记为 $pattern=\cdots\$\$\$\$a_0a_1a_2\cdots a_{L-1}$,其中 \$ 为假想字符(即 $a_i=\$,i<0$)。

对于字符串 $Q=q_0q_1\cdots q_n$ 和 $R=r_0r_1\cdots r_n$,定义它们为相一致($unify$),只要满足 $q_i=r_i$ 或 $q_i=\$$ 或 $r_i=\$$ $(i>0)$,同时 $q_0\neq r_0$ 或 $q_0=\$$ 或 $r_0=\$$。直观地说,就是首字符不同(或者为假想字符 \$),其余字符相同(或者为假想字符 \$)。

Delta2 表的第 j 项的值,$D_2(j)=L-k$,要求 k 满足串 $a_ka_{k+1}a_{k+2}\cdots a_{k+L-j-1}$ 和 $a_ja_{j+1}\cdots a_{L-1}$ 为相一致,同时不存在串 $a_ia_{i+1}a_{i+2}\cdots a_{i+L-j-1}(k<i<j)$和 $a_ja_{j+1}\cdots a_L$ 为相一致。直观地说,k 是最右能和 $a_ja_{j+1}\cdots a_L$ 为相一致的子字符串的起始位置。

定义完 Delta1 和 Delta2 表后,BM 的算法流程可以用图 5.7 描述。

5.4.3 HBM 算法概述

传统 BM 算法是被验证为相当有效的一种单模匹配算法。但是在使用 BM 算法进行多模匹配的过程中,需要对其进行很多改进。主要体现在这几个方面。

(1) 每次比较的单位由单字符变为若干字符组成的字符块。

(2) 比较方式由字符的对应比较,变为查找字符块在 Delta1 表(字符块信息的启发式移位表)中的值是否满足一定条件。

(3) Delta2 表的构造和使用完全不同。

(4) Delta1 表急剧增大,需要特定的机制减小其空间开销。

这四点改变,人们通过简单的观察就能发现。

观察 A: 如果比较的单位仍然为单字符,由于特征字符串数目很多,那么每个字符都

```
1. 构造Delta1 表;
   构造Delta2 表;
   L←待比较字符串string 长度;
   i←特征字符串pattern 长度;
2. if i>L then return false.
   end if
3. j←特征字符串长度;
4. if j = 0 then return i+1
   end if
5. if string(i) = pattern(i)
   then
     i←i−1;
     j←j−1;
     goto 4
   else
     i←i+max{D1(string(i)),D2(j)};
     goto 2
   end if
```

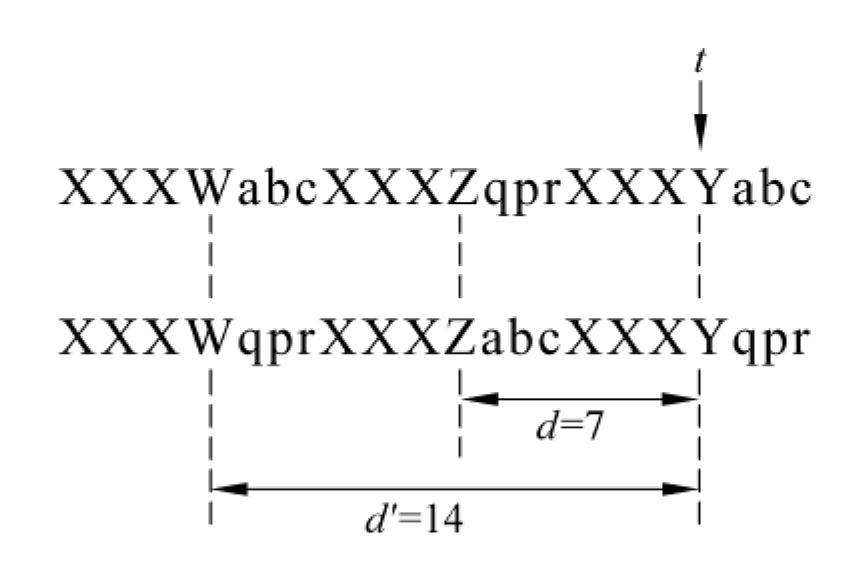
图 5.7　BM 的算法流程

很可能在特征字符串最右端出现,此时其在 Delta1 表的对应项为零。如果有上百个特征字符串,那么 Delta1 表中为零项有接近上百项,而 Delta1 表的大小固定等于字符表的大小,若以单字节表示一个字符来说,Delta1 表只有 256 项。此时,绝大部分的 Delta1 表项均为 0,算法失效。

观察 B:如果比较方式仍然为字符与字符的对应比较,那么每次比较必须和所有特征字符串的对应位置上的字符或字符块进行比较。需要的 I/O 操作和比较操作次数等于特征字符串数,显然效率过低。

观察 C:Delta2 表的构造如果沿用 BM 算法,则算法出错。例如,图 5.8 中,假设比较单位的字符块大小为 2,有 2 个长度为 21 字符的特征字符串,如果采用 BM 算法构造 Delta2 表,则位置 $t=17$ 的值 $D_2'(t)=17$,而采用后文提到的 HBM 算法得到 $D_2(t)=10$。输入字符串如图 5.8 所示,当第一轮比较在位置 t 失败时,如果将比较指针移动 $D_2'(t)$ 距离后(如图 5.8 中第 4 行所示),则移动距离过大,跳过了本可以匹配的字符串部分;而如果将比较指针移动 $D_2(t)$ 后,如图 5.8 中第 5 行所示,能够找到正确的匹配。

观察 D:将字符块作为基本单位时,Delta1 表的大小就会急剧膨胀。原始的 BM 算法需要 $2^8=256$ 项,而以 2,3,4 个字符的字符块作为基本比较单位的话,则分别需要



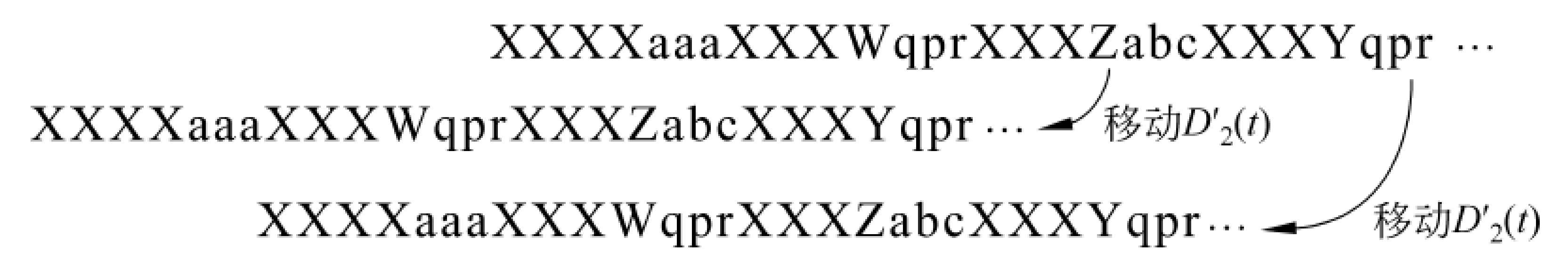


图 5.8 Delta2 表构造方式差异的说明

$2^{16}=64K$，$2^{24}=24M$，$2^{32}=4G$ 项。对于这个问题，一些算法如 WM 和 RSI 等，采取了回避的态度，其前提是特征字符串数目不多，采用 2 个字符的字符块能保证算法有效的情况下，直接存放 64K 项 Delta1 表。另一些算法如 Setwise BMH 等，通过组织特定的树结构，存放 Delta1 表，但是每次查表，往往需要多次的 I/O 操作，在实现上有局限性。我们提出的 HBM 算法则是通过可碰撞的散列表来组织 Delta1 表，每个字符块在 Delta1 表中有不再唯一对应一项，而是把字符块的散列值作为其在 Delta1 表中的索引位置。

根据以上观察，可以了解 HBM 作为多模匹配算法的主要特征如下。

(1) 每次比较的基本单位是若干个字符(比如 4 个字符)组成的字符块。

(2) 每次对一段子字符串的匹配是从右向左进行，一旦某个字符块比较失败则不用比较剩下的字符块，而是根据启发式移位表 Delta1 表和 Delta2 表向右移动比较指针。

(3) 算法使用两张启发式移位表：Delta1 表和 Delta2 表，充分利用了蠕虫特征码的信息和比较过的字符块信息。

(4) 当某个子字符串比较成功后，还需要和蠕虫特征码进行精确比较，才能确定其为特征码之一，或是一次误判(false-positive)。

HBM 算法主要包括初始化工作和运行时工作两部分。初始化工作主要由控制层面完成对特征码的初始化处理，组织两张启发式移位表。

运行时工作主要由完成网络网包的接收、分类、扫描和转发功能。

5.4.4 HBM 算法的初始化流程

1. Delta1 表的初始化构造

Delta1 表构造的算法如图 5.9 所示。算法寻找每个字符块在特征字符串最右出现位置，据此计算出“匹配失败”时至少可以将指针向右移动的距离，并将这个“移位距离”记录

在 Delta1 表中。和原 BM 算法相比，它有以下不同之处。

（1）比较的基本单位变为字符块。

（2）字符块在 Delta1 表中的索引位置由散列函数值决定。

（3）如果两个字符块由于散列冲突，对应 Delta1 表的同一项，则此项存放两者中的较小“移位距离”。

```
1. ∀x,d1(x)=L−b+1
2. for i:=0 to n−1
     for j:=0 to L−b
               x=h(s_ij s_i(j+1) ··· s_i(j+b));
       if d1(x)>L−b−j  then  d1(x)=L−b−j
       end if;
     end loop j;
   end loop i;
```

图 5.9　Delta1 表构造算法描述

Delta1 表的使用也很方便。在对输入子字符串的某轮匹配中，记 j 为已经比较成功的字符块数（即比较最右字符块时 $j=0$，最右边第 2 个字符块时 $j=1$，以此类推）。判断某次字符块比较是否成功，只要将此字符块在 Delta1 中的值 D_1 和 j 进行比较。如果 $D_1\leqslant j$，则比较成功，将比较指针左移移位，继续下一个字符块比较；如果 $D_1>j$，则比较失败，此轮匹配也失败，可以将比较指针向右移动 D_1 距离，开始下一轮的匹配。

需要注意的是，在 HBM 算法中，每次比较 Delta1 表值时，使用“⩽”而不是“＝”。这是由多个特征字符串的特性带来的，和是否采用散列方式组织 Delta1 表无关。例如，图 5.10 给出了 $L=8$，$b=2$ 的一个例子，有 2 个特征字符串“xxxxxxcd”和“xxxxxcda”。由于字符块“da”和“cd”都是在特征字符串最右位置，因此 $D_1(cd)=D_1(da)=0$。现在输入字符串的前 L 个字符和特征字符串 2 完全一致，匹配过程如下，第一次比较满足 $D_1(da)\leqslant 0$，比较成功；第二次比较 $D_1(cd)=0\leqslant 1$，此时 $D_1(cd)\neq 1$，如果使用，则比较失败，显然不合理。这就是为何算法使用“⩽”而不是“＝”。

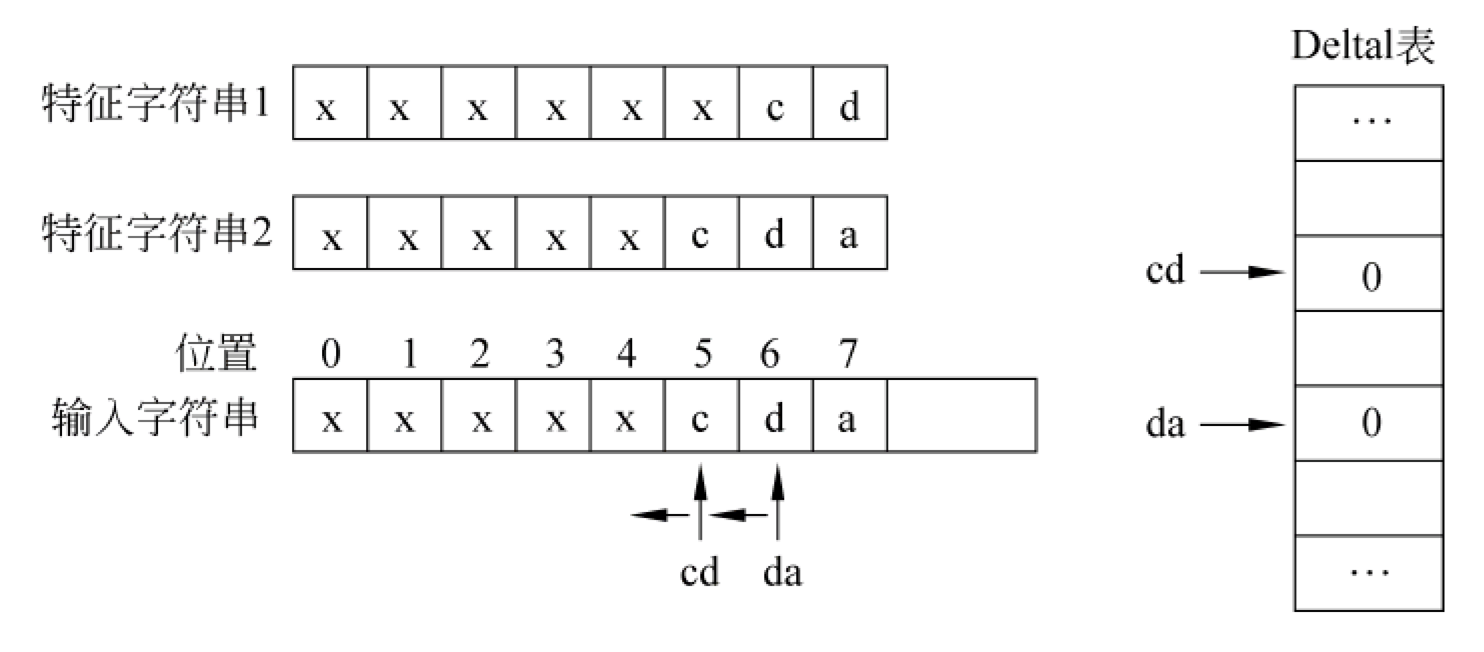


图 5.10　Delta1 表使用举例

2. Delta2 表的初始化构造

Delta2 表构造的基本思想也来自 BM 算法，在匹配失败时，根据之前比较过的子特征字符串的信息，得到至少可以移动的距离。介于在多模匹配下，Delta2 表比 Delta1 表的

移植困难很多，它在很多算法(如 WM、Setwise BMH、RSI 等)中均被忽略，而极少有人提及。本节将给出一种 Delta2 表的构造和使用方法，作为 HBM 算法的一部分。由于和原 BM 算法的 Delta2 表大相径庭，故不再给出本算法的改进说明。

为了便于描述，将任意字符串 $pattern = a_0a_1a_2\cdots a_{L-1}$，扩充记为 $pattern = \cdots \$\$\$\$a_0a_1a_2\cdots a_{L-1}$，即 $\forall i<0, a_i=\$$。其中 \$ 为假想字符，它和任何字符相等，同时任何含有 \$ 的字符块，其 D_1 值为 $D_1=0$。这里对所有特征字符串都作了这种扩充。

在给出 Delta2 表的构造和使用之前，先给出某个子字符串“相一致”，以及函数 rpr (rightmost plausible reoccurrence)的定义。

“相一致”指的是特征字符串中，某个子字符串和最右端的子字符串，它们的首字符块有不同的特性，其余对应字符块则有共同的特性。根据 5.4.1 节的讨论，可以发现：特征字符串的最后几个字符块的 D_1 值满足条件 5-5：

$$D_1(a_{L-b}\cdots a_{L-1}) \leqslant 0, D_1(a_{L-b+1}\cdots a_{L-2}) \leqslant 1, \cdots \tag{5-5}$$

因此，相一致的子字符串也要求满足上述条件。同时，因为两者的首字符块特性不同，所以其首字符块在 Delta1 表中的值要求满足 $D_1>k-1$，其中 k 为子字符块的长度。形式化的定义如下。

定义特征字符串中的子字符串 $C=c_0c_1\cdots c_n$“相一致”，要求满足条件 5-6：

$$\begin{cases} \forall 1 \leqslant i \leqslant n-b+1, D_1(c_ic_{i+1}\cdots c_{i+b-1}) \leqslant n-b+1-i \\ D_1(c_0c_1\cdots c_{b-1}) > n-b+1 \text{ 或 } c_0 = \$ \end{cases} \tag{5-6}$$

其中，$D_1(c_ic_{i+1}\cdots c_{i+b-1})$为字符块 $c_ic_{i+1}\cdots c_{i+b-1}$ 在 Delta1 表中的移位值。

定义函数 $y=rpr(S_i,t)$，$t\in[0,L-b-1]$，要求在特征字符串 S_i 中，从位置 t 往左(不包括 t)，找到长度为 $L-t$ 的最右的相一致的子字符串，记此子字符串为位置 k 开始的 $s_{ik}s_{i(k+1)}\cdots s_{i(k+L-t-1)}$，那么 $rpr(S_i,t)=k+1$。算法描述如图 5.11 所示。

图 5.11 中为了下面表述方便，记函数 $D_1'(i,t)$ 为取字符块 $s_{it}s_{i(t+1)}\cdots s_{i(t+b-1)}$ 的在 Delta1 表的值，即 $D_1'(i,t)=D_1(s_{it}s_{i(t+1)}\cdots s_{i(t+b-1)})$。

在计算 rpr 函数之后，可以据此来构在 Delta2 表。Delta2 表长度固定为 $L-b+1$，记 Delta2 表的第 t 项的值 $D_2(t)$，则其值由式(5-7)求得：

$$D_2(t) = L-b+1-\max_r\{rpr(S_r,t)\} \tag{5-7}$$

其中，$0\leqslant t\leqslant L-b-1$，$\max_r\{rpr(S_r,t)\}$表示对于 $r=0,1,\cdots,n-1$，取这 n 个 rpr 值中的最大值。

为了保持整体算法的简洁性，定义 $D_2(L-b)=1$，也就是 $rpr(S_r,L-b)=L-b$。这样定义的原因是，可以保证在使用 $D_2(L-b)=1$ 时，必然会有 D_2 值小于 D_1 值，也就是 $D_2(L-b)=1$ 不会被使用，因而对算法没有影响。图 5.12 为 Delta2 表构造的算法描述。

由于 Delta2 表的构造和使用比较复杂，其正确性的详细证明将在 5.4.7 节中分析，本节给出了一个例子加以说明。

```
1. if t=L−b then return L−b
2. k←t
3. if k≤−(L−b−t) then return k
   end if
4. if k > 0 then goto 5
            else goto 6
   end if
5. if(∀0≤r<L−b−t, D'1(i,k+r)<L−b−t−r)and D'1(i,k−1)>L−b−t
   then return k
   else goto 7
   end if
6. if ∀r,0≤r<L−b−t+k,D'1(i,r)<L−b−t+k−r
     then return k
     else goto 7
   end if
7. k←k−1
   goto 3
```

图 5.11　函数 $rpr(S_i, t)$ 计算的算法描述

```
1. 构造Delta1 表
2. ∀t , D2(t)=Infinite_max
3. for i:=0 to n−1
     for t:=0 to L −b
       d←L−b+1−rpr(Si,t);

       if D2(t)>d then D2(t)←d
       end if;
     end loop t ;
   end loop i ;
```

图 5.12　Delta2 表构造的算法描述

如图 5.13 所示，在 $L=8, b=2$ 的情况下，假设所需要的 D_1 值已经由图 5.13 中的 Delta1 表给出，那么对于这个特征字符串 S，就可以算出其 $rpr(S,t)$ 值。并且，假设特征字符串 S 的各个 rpr 表项值均为对应的最大值，还可以计算出了 Delta2 表的值。

3. 特征码散列表初始化构造

由于 HBM 算法需要在成功匹配时进行精确比较，则合理组织待匹配的特征码，减少访问时的 I/O 开销是必要的。引擎实现中，将特征码组织成散列链表方式，大大减少了 I/O 访问次数，提高了效率。

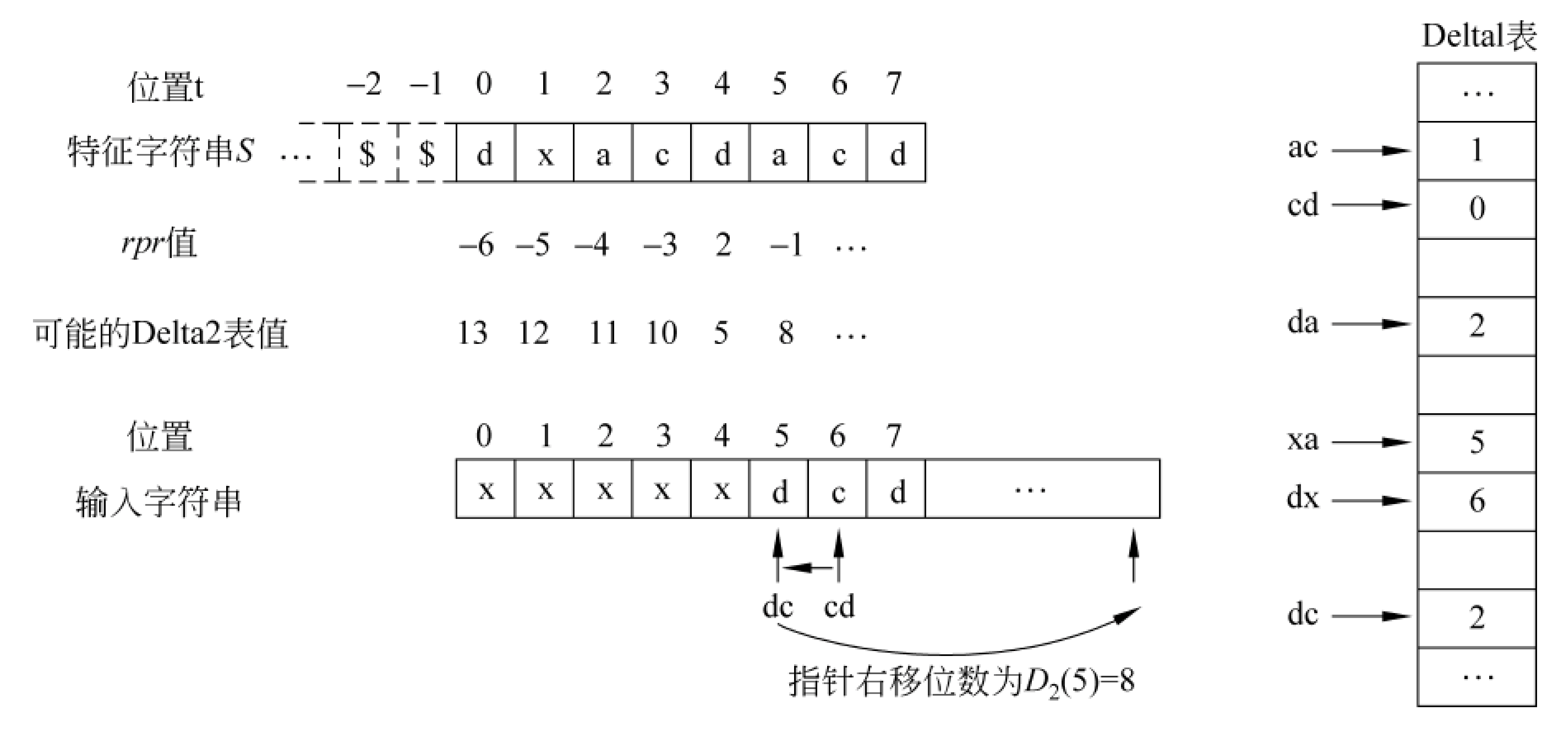


图 5.13 Delta2 表的构造和使用举例

其示意图如图 5.14 所示。

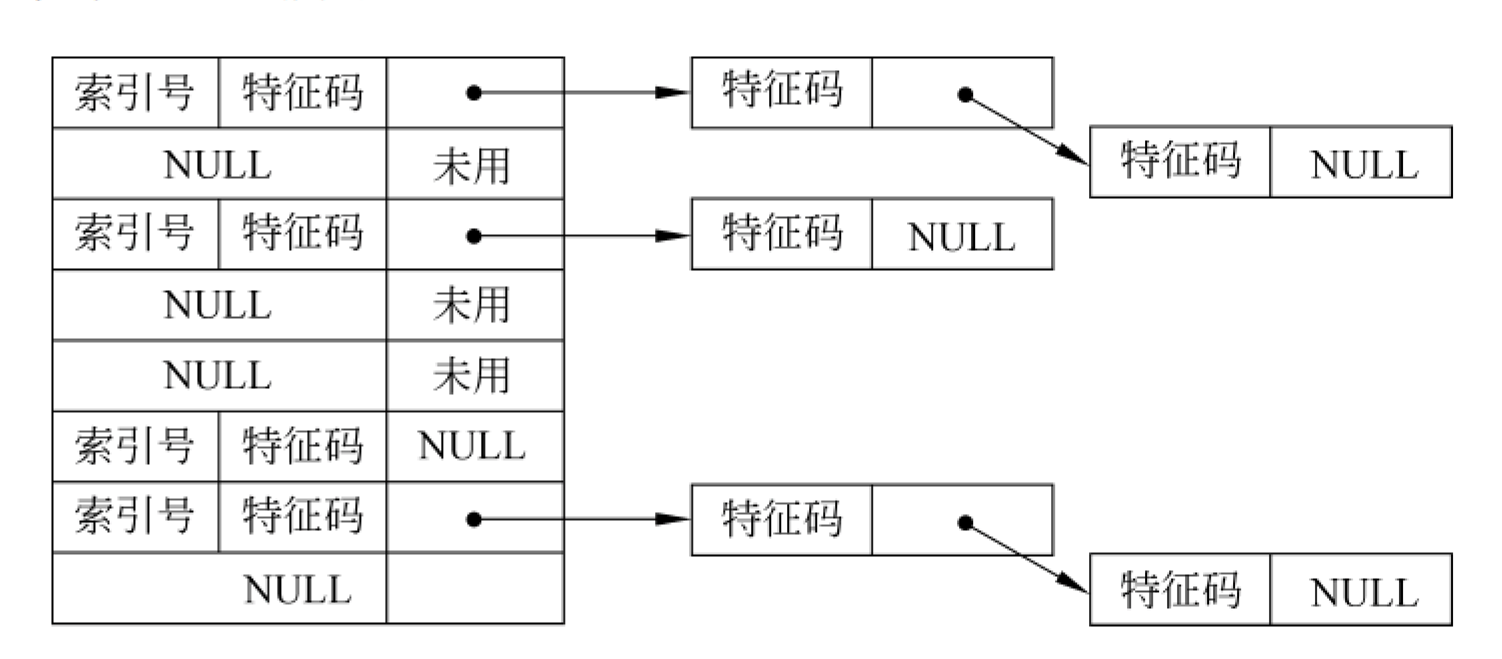


图 5.14 散列链表结构示意图

在实现散列存储的时候，也需要一个散列函数 $h_s(S_k)$ 对每个特征码计算索引号，此散列函数可以是一些标准的散列计算方法(如 MD5 等)，也可以是简单的逻辑运算从而加快处理速度。算法描述如图 5.15 所示。

```
1. for  i:=0 to n−1
      entry←h_s(S_i)
      if 索引号为entry项未用
      then 加入S_i，指针标记改为NULL
      else 在index处的链表尾加入S_i
     end if
   end loop i;
```

图 5.15 特征码的散列链表构造

5.4.5　HBM 算法运行流程

记输入的字符串为 String，记第 i 个字符为 $str(i)$，$i=0,1,\cdots$；记从第 i 个字符开始的字符块在 Delta1 表中的值为$\overline{D_1(i)}$，即$\overline{D_1(i)}=d_1(h(str(i),str(i+1),\cdots,str(i+b-1)))$。

运行时的算法流程描述如图 5.16 所示。

```
1. 构造Delta1 表;
   构造Delta2 表;
   i←L−b+1
2. if i > sizeof(string) then return search failed
   end if
3. j←0
4. if D1(i) > j then goto 9
   end if
5. if j≥L−b then goto 7
   end if
6. i←i−1;
   j←j+1;
   goto 4
7. Precisely Comparison of str (i)str(i+1)···str (i+L−1)
   if Match then return search succeeding
   end if
8. i←i+L−b+2 ;
   goto 2
9. i←i+max{D1(i),D2(L−b−j)};
   goto 2
```

图 5.16　HBM 运行时算法流程描述

在上述算法流程描述中，第 3 步至第 6 步表示对输入的子字符串的一轮匹配。当第 4 步的条件不满足时，表示此次比较成功，可以继续比较当前位置左侧的字符块；当第 4 步的条件满足时，说明本次比较失败，因而本轮的匹配也失败，可以将比较指针 i 向右移动 $\max\{\overline{D_1(i)},D_2(L-b-j)\}$位，然后再继续对新的子字符串进行下一轮匹配；当第 5 步的条件满足时，表示输入子字符串的每个字符块均比较成功，也就是本轮匹配成功。但是有可能出现“伪匹配成功”(也称为误判，false-positive)的情况，还需要和特征字符串进行精确比较，才能最终确定是否真正匹配成功(见第 7 步)。

伪匹配成功的发生，是由多模匹配（多特征字符串）本身特性带来的，和是否采用散列方式无关，下面举例说明。当 $L=4,b=2,n=3$ 的情况下，3 个特征字符串的分别是"$xxab$"、"$xxbc$"和"$xxcd$"，所以字符块"ab"、"bc"和"cd"在 Delta1 表中的值 $D_1(ab)=D_1(bc)=D_1(cd)=0$。因而，当某一轮的输入子字符串为"abcd"时，此轮的每次比较都将成功，但是子字符串"abcd"却不是任何一个特征字符串，本轮匹配是一次伪匹配成功。

此外，当发生伪匹配成功时，如果一一比较所有的特征字符串，那么 I/O 访问操作的开销将会很大。因此，可以将所有特征字符串以散列链表方式存放，从而减少精确比较时需要进行的 I/O 访问操作。这是对于上述算法第 7 步精确比较的一点补充。

5.4.6 HBM 算法在多核平台上的优化

将 HBM 算法在多核平台上做了实现，充分利用了多核的体系结构特点优势，从而达到了较好的吞吐率性能。

分层的存储结构有利于发挥散列算法的特性，减少 I/O 操作的时间开销。可以将 HBM 算法中的 Delta1 表和 Delta2 表放入缓存中，因此算法的 I/O 操作的时间开销可以大大降低。如果不使用散列方式的算法，即使是以 2 个字节作为字符块，也需要至少 16K 大小的 Delta1 表，无法放入处理器本地内存，只能存放于更慢速的 SRAM 或 DRAM，从而增加了系统的 I/O 开销。

HBM 算法中没有复杂的数学计算运算，比较适合通用多核平台的指令结构。由于通用多核平台的优势在于网络网包处理过程，它并没有提供复杂的通用计算指令，而 HBM 算法可以将散列运算用与或操作完成，不需要复杂的乘除指令支持，很好地适应了通用多核平台。

1. 功能模块图

图 5.17 给出了多核平台上实现的匹配引擎 Demo 的功能模块图。网络网包接收模块（Packet RX）从以太网接口上收到的网包内容存入 DRAM，将网包信息通过一个 Scratch Ring 传给 HBM 过滤模块，过滤模块根据网包信息，从 DRAM 中取出网包载荷内容并进行扫描，将通过检测的正常网包信息通过另一个环的结构传给转发模块（Packet TX），而对于包含蠕虫特征代码的危险网包，过滤模块进行记录并发出警告。

2. 存储单元的分配

如表 5.2 所示，将 HBM 算法中的 Delta1 表和 Delta2 表放入缓存中，特征字符串集合则存放在 DRAM 存储器中。在 DRAM 中，除了保留一份原始特征字符串集合的副本外，将这些特征字符串组成散列链表的方式，这样在进行精确比较的时候可以有效减少 I/O 操作次数。

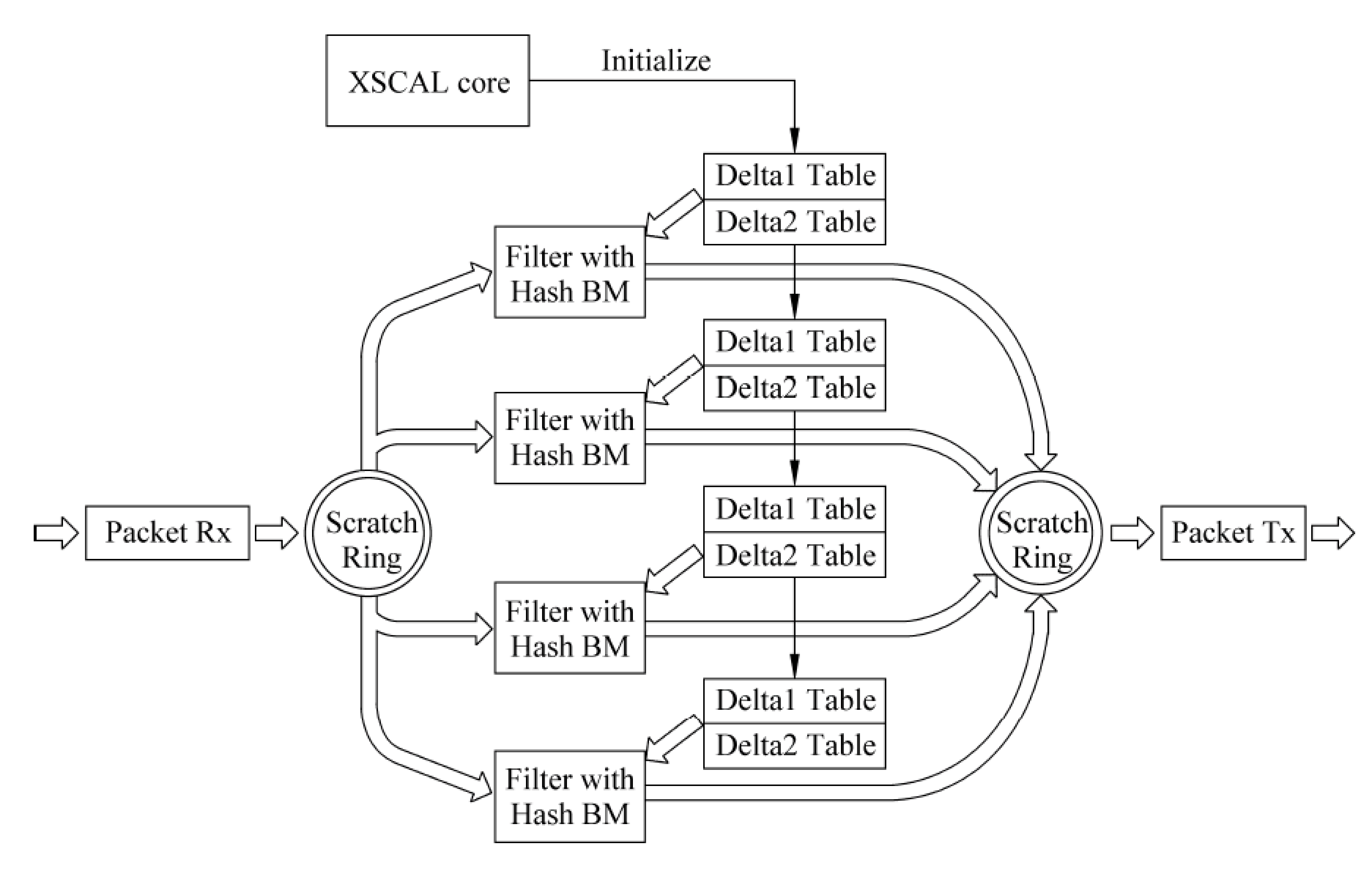


图 5.17　HBM 算法在多核通用多核平台平台上的功能模块图

表 5.2　各数据结构在系统中的存放位置

存放内容	存放空间	存取速度
Delta1 表	Cache	快
Delta2 表	Cache	快
原始特征字符串集合	DRAM 中顺序存放	慢
结构化的特征字符串集合	DRAM 中散列链表方式存放	慢

3.　优化小结

综上所述，本章在上实现 HBM 多模匹配算法时，主要的优化如下。

（1）将 Delta1 和 Delta2 这两张访问频率较高的表，存放在 I/O 存取速度快的存储单元，而对于访问频率低的特征字符串集合存放在存取速度慢的单元。

（2）将特征字符串组织成散列链表的方式，减少精确匹配时的 I/O 操作次数。

（3）对于输入网包的载荷进行预读取功能，减少 I/O 操作次数。由于无论读取字节的多少，对于 DRAM 的一次 I/O 操作所花费的时间相近，过滤引擎对于输入的网包载荷每次预读入最大的 64 个二进制字符。在一轮匹配失败，需要移动比较指针时，判断目的指针位置的字符块，是否已经超过预读入的最后一个字符块。如果没有超过，则依然可以从上次预读入的字符中得到本轮匹配的子字符串，省去了一次 I/O 操作，如果超过，则必

须重新读入。

5.4.7 HBM 算法的证明与分析

1. HBM 算法证明

HBM 算法的正确性，主要包括 Delta1 表的正确性、Delta2 表的正确性和精确比较部分的正确性三部分。其中，精确比较部分的正确性是显而易见的，而 Delta1 表基本沿用了 BM 算法的思路，只是使用散列函数加以改造，也比较简单。因此，这里主要讨论 Delta2 表的正确性的证明。

对于 Delta2 表的正确性，本章采用反证法证明。在上述算法中需要用到 Delta2 表的值的只有在第 9 步。在第 9 步或第 7 步移动指针 i 之前，本轮的是在判断字符串 String 中第 $i+j-L+b-1$ 个的字符到第 $i+j+b-1$ 个的字符是否为特征字符串。同时，当第 9 步需要移动指针 i 时，如果 Delta1 表项值大于 Delta2 表项值，则算法的进行和 Delta2 表无关。为了证明 Delta2 表的正确性，不妨再假设此时 Delta2 表项值较大。

证明：

反设第 9 步 $i=i_0$，此时 $i \leftarrow i_0+D_2(L-b-j)$ 移动的距离过大，导致错误。即 $\exists r, 0<r<D_2(L-b-j)$，$str(i_0+r+j-L+b-1), \cdots, str(i_0+r+j+b-1)$ 完全等于某个特征字符串 S_k。Delta2 表的正确性证明示意图如图 5.18 所示。

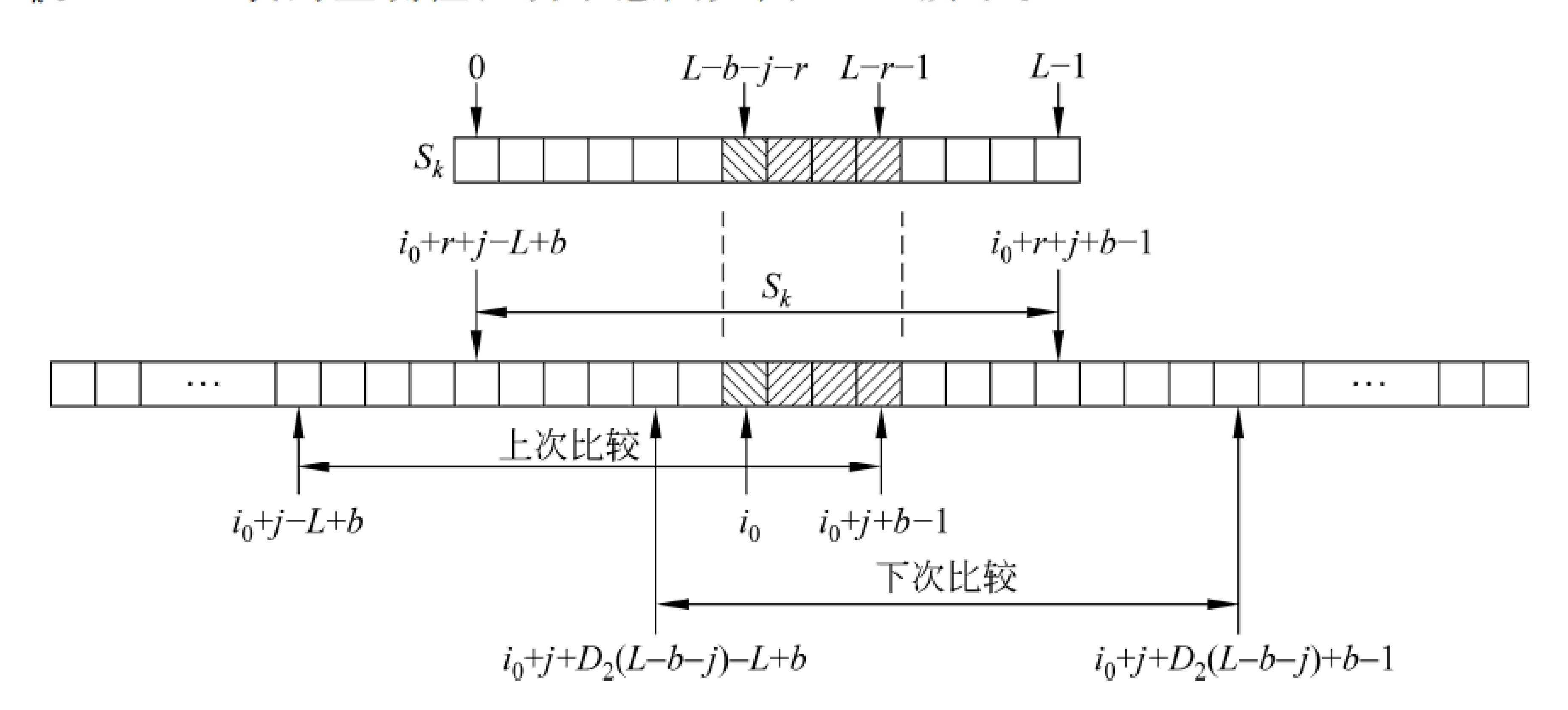


图 5.18 Delta2 表的正确性证明示意图

由假设可知，输入字符串的 i_0 到 $i_0+j+b-1$ 位置字符，相当于特征字符串 S_k 中的 $L-b+1-j-r$ 到 $L-r$ 位置的字符。由于此时执行至算法第 9 步产生 i_0，$\forall t, L-b-j-r<t\leqslant L-r-1$（即图 5.18 阴影部分为"相一致"的子字符串），满足条件式(5-8)。

$$\begin{cases} d_1(k,t)<L-r-t \\ d_1(i,L-b-j-r)>j-b \end{cases} \tag{5-8}$$

所以，计算 rpr 函数：

$$rpr(S_k, L-b-j) \leqslant (L-1)-(L-r-1)=r \tag{5-9}$$

从而，得到 Delta2 值满足：

$$D_2(L-b-j) \leqslant rpr(S_k, L-b-j) \leqslant r \tag{5-10}$$

与假设的 $r<D_2(L-b-j)$ 矛盾，原假设不成立，Delta2 表构建的正确性证毕。

2. HBM 算法解析分析

本节主要讨论 HBM 算法的效率，包括如下。

（1）给出 Delta1 表平均移位距离的计算方式，即给出此算法的时间性能的主要计算方法。

（2）给出 Delta1 表的平均移位距离和大小的关系，能够合适选择存储容量来达到较好的时间性能。

（3）给出使用 Delta2 表的概率，可以看到 Delta2 表的使用效率不高。

（4）给出伪匹配概率，即给出了额外的时间开销计算。

1）平均移位距离分析

HBM 算法的移位距离是指每当比较失败后算法的平均移位距离，它主要由 Delta1 表和 Delta2 表中值决定。

一方面，在 HBM 算法中，每次的字符块的比较，是通过将 Delta1 表项的值和当前移动位置 j 来实现的(见算法第 4 步)。因此，如果 Delta1 表的平均移位距离很小，则通过这种 Hash 方式比较，“成功”的概率很大，需要比较次数增加，从而大大降低了算法的效率。

另一方面，在 HBM 算法中，当一次字符块的比较失败后，可以向后移动指针的距离也是由 Delta1 表和 Delta2 表两者共同决定的。

因此，HBM 算法中平均每个字符消耗的时间为

$$\begin{aligned}\overline{T_{\text{char}}} &= t_{\text{comp}} \cdot \overline{c_{\text{char}}} + \frac{t_{\text{exact-comp}} \cdot P_{\text{false-positive}}}{\overline{\text{distance}_{\text{comp}}}} \\ &= \frac{t_{\text{comp}} + t_{\text{exact-comp}} \cdot P_{\text{false-positive}}}{\overline{\text{distance}_{\text{comp}}}}\end{aligned} \tag{5-11}$$

其中，t_{comp} 是一次字符块比较所消耗的时间，$\overline{c_{\text{char}}}$ 是对于每个字符平均需要的比较次数，$t_{\text{exact-comp}}$ 是精确比较需要消耗的额外时间，$P_{\text{false-positive}}$ 是误判概率，$\overline{\text{distance}_{\text{comp}}}$ 每当比较失败后算法的平均移位距离。

因为 t_{comp} 和 $t_{\text{exact-comp}}$ 是受系统 I/O 和数据结构影响的常量值，所以 $P_{\text{false-positive}}$ 和 $\overline{\text{distance}_{\text{comp}}}$ 就成为影响系统性能的主要参数。

从 HBM 算法描述中可以看出，Delta2 表的构造需要用到 Delta1 表的表项值，其表

项值受到 Delta1 的平均表项值——移位距离的影响。而且，本章将在接下来的几节中进行解析分析，可以得出 Delta2 表项值的使用概率比 Delta1 表多，即 Delta2 表对算法平均移位距离的影响相对于 Delta1 表而言低很多，并给出 Delta1 表的平均移位距离计算、Delta1 表的空间大小选择、整体算法的平均移位距离等。

（1）Delta2 表的使用概率。

这里指的 Delta2 表的使用概率是当发生移位时，采用 Delta2 表而不采用 Delta1 表的值的概率，也就是此时 $D_2>D_1$。对于特殊情况 $D_2=D_1$，也就是当发生移位时，Delta1 和 Delta2 表的移位距离相等，使用哪个表的值相同，此时假定使用 Delta1 表的值（因为，这种情况下，如果没有 Delta2 表存在，依然可以仅依靠 Delta1 表得到最大的移位距离）。

从仿真实验中，也可以看到移位时，使用 Delta2 表的概率的确远小于使用 Delta1 表的概率。Delta2 表的使用概率很难求得精确解，下面尝试给出 Delta2 表的使用概率的近似范围。

从仿真实验发现，Delta2 表的最后几项很可能出现 $D_2(L-b-j)=j+1$，例如，某次情况中，$L=32$，$b=4$，$n=512$，$m=640\times6=3840$，Delta2 表产生情况如表 5.3 所示。

表 5.3 Delta2 表举例

$L-b-j$	0	…	19	20	21	22	23	24	25	26	27	28
$D_2(L-b-j)$	39	…	23	16	8	10	6	5	4	3	2	1

此时，如果移位发生在最后 6 个位置时，$D_2\leqslant D_1$，都不可能使用 Delta2 表。如果要使用 Delta2 表，则必须首先满足 $D_2(L-b-j)>j+1$，如表 5.3 中 $L-b-j=22$ 的情况，此时 $D_2(22)=10$，才有可能用到 Delta2 表。现在根据这条思路，给出了表 5.4，来计算 Delta2 表使用的概率的上下界。

表 5.4 Delta2 表使用的概率的上下界计算

比较成功次数 j 次时发生移位	此时发生移位的概率	此时使用 Delta2 表的概率
$j=0$	$P(j=0)=a$	0
$j=1$	$P(j=1)=(1-a)\cdot a^2$	$p_1\cdot q_1$ 和 p_1 之间
$j=2$	$P(j=1)=(1-a)\cdot(1-a^2)\cdot a^3$	$p_2\cdot q_2$ 和 p_2 之间
…	…	…
$j=k$	$P(j=k)=(1-a)\cdot(1-a^2)\cdots$ $(1-a^k)\cdot a^{k+1}$	$p_k\cdot q_k$ 和 p_k 之间

其中，$k=L-b$，$a=\left(1-\dfrac{1}{m}\right)^n$；$p_k$ 表示发生移位时，Delta2 表的值 $D_2(L-b-j)>j+$

1 的概率；q_k 表示发生移位时，Delta1 表的值 $D_1(x)=j+1$ 的概率。综上可知：

① 当使用 Delta2 表中的移位值时，必然有 Delta2 表的值 $D_2(L-b-j)>j+1$，即 $\sum_{i=1}^{k} P(j=i)\cdot p_k$ 是使用 Delta2 表中的移位值的概率的上界。

② 当 Delta2 表的值 $D_2(L-b-j)>j+1$，并且 Delta1 表的值 $D_1(x)=j+1$ 时，$D_2>D_1$，必然使用 Delta2 表中的移位值，即 $\sum_{i=1}^{k} P(j=i)\cdot p_k\cdot q_k$ 是使用 Delta2 表中的移位值的概率的下界。

所以，Delta2 表的使用概率在 $\sum_{i=1}^{k} P(j=i)\cdot p_k\cdot q_k$ 和 $\sum_{i=1}^{k} P(j=i)\cdot p_k$ 之间。

这里，没有出现特征字符串长度 L 和字符块大小 b，原因是当 k 较大时，$P(j=k)$ 很小，非常接近 0，Delta2 表的使用概率主要取决于上述多项式中靠前的几项。因此，特征字符串达到一定长度后，Delta2 表的使用概率基本是相同的。

$P(j=k)$ 的计算比较简单，在此不做展看讨论了。

对 q_k 的计算，由于移位是发生在 $j=k$ 的情况下，必然有 $D_1(x)\geqslant j+1$，则 $D_1(x)=j+1$ 的概率相同，即 $q_1=q_2=\cdots=q_k$，记其为 q，则 $q=1-\left(1-\frac{1}{m}\right)^n=1-a$。

对 p_k 的计算，要得到 $D_2(L-b-j)>j+1$，必须使得所有特征字符串中，从 $L-b-j-1$ 开始的字符串不满足相一致（就是当前位置向左一位开始的字符串不满足相一致）。所以，

$$p_1=\left\{1-\left[1-\left(1-\frac{1}{m}\right)^n\right]\cdot\left[1-\frac{1}{m}\right]^{2n}\right\}^n=[1-(1-a)\cdot a^2]^n;$$

$$p_2=\left\{1-\left[1-\left(1-\frac{1}{m}\right)^n\right]\cdot\left[1-\left(1-\frac{1}{m}\right)^{2n}\right]\cdot\left[1-\frac{1}{m}\right]^{3n}\right\}^n$$
$$=[1-(1-a)\cdot(1-a^2)\cdot a^3]^n;$$

$$\vdots$$

$$p_k=[1-(1-a)\cdot(1-a^2)\cdots(1-a^k)\cdot a^{k+1}]^n;$$

下面举例解释如下，例如 $p_2=\left\{1-\left[1-\left(1-\frac{1}{m}\right)^n\right]\cdot\left[1-\left(1-\frac{1}{m}\right)^{2n}\right]\cdot\left[1-\frac{1}{m}\right]^{3n}\right\}^n$。

考察某个特征字符串 S_t，从 $L-b-j-1$ 开始的字符串是否相一致。首先，对于最后一个字符块，$\left(1-\frac{1}{m}\right)^n$ 表示所有特征字符串的最后一个字符块和其不发生散列碰撞的概率，也就是其 D_1 值 $D_1>0$ 的概率，则 $\left[1-\left(1-\frac{1}{m}\right)^n\right]$ 表示其 D_1 值满足 $D_1\leqslant 0$ 的概率；同理 $\left[1-\left(1-\frac{1}{m}\right)^{2n}\right]$ 表示倒数第二个字符块，其 D_1 值满足 $D_1\leqslant 1$ 的概率；$\left[1-\frac{1}{m}\right]^{3n}$ 表示

倒数第三个字符块，其 D_1 值满足 $D_1>2$。因此，$\left[1-\left(1-\frac{1}{m}\right)^n\right]\cdot\left[1-\left(1-\frac{1}{m}\right)^{2n}\right]\cdot\left[1-\frac{1}{m}\right]^{3n}$ 表示 $L-b-j-1=L-b-3$ 开始的字符串满足相一致，也就是对于这个特征字符串 S_t，$D_2(L-b-j)=j+1=3$。

所以，p_2 表示对于所有的 n 个特征字符串，均不满足 $D_2(L-b-j)=j+1$（也就是 $D_2(L-b-j)>j+1$）的概率。在此情况下，Delta2 表中存放的 D_2 值为 n 个特征字符串算出的每个 D_2 值中最小的，那么同样有 $D_2(L-b-j)>j+1$。

由以上分析知，Delta2 表的使用概率在 $\sum_{i=1}^{k}P(j=i)\cdot p_k\cdot q_k=q\cdot\sum_{i=1}^{k}P(j=i)\cdot p_k$ 和 $\sum_{i=1}^{k}P(j=i)\cdot p_k$ 之间，如图 5.19 所示（$n=512$）。

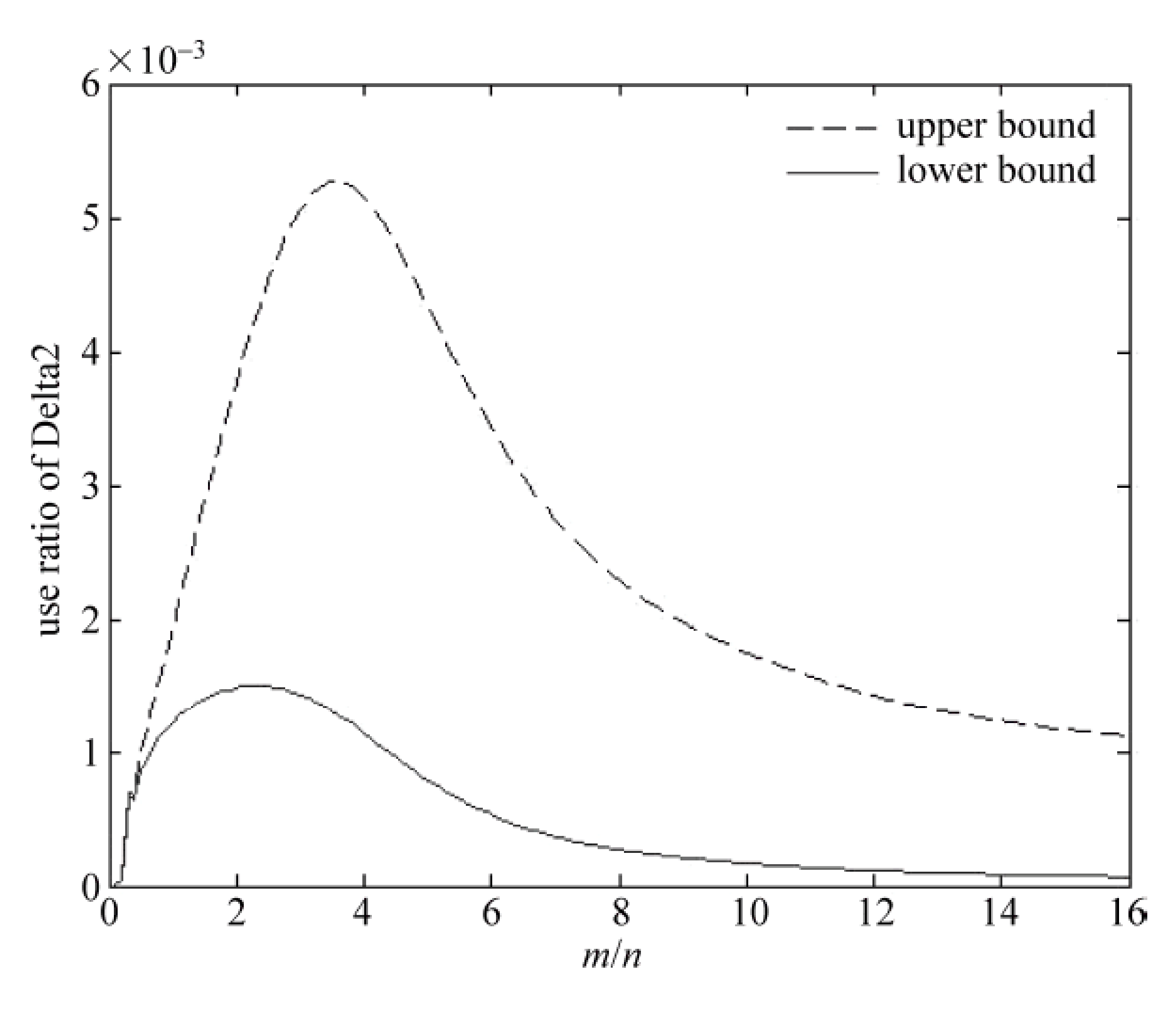


图 5.19　$n=512$ 时，Delta2 表使用概率与 m/n 的关系图

由图 5.19 可知，当 m/n 增大时，Delta2 表的使用概率先随之增加，再随之降低。起先增加的原因是，当 Delta1 表增大时，散列碰撞减少，Delta2 表的平均移位距离增加，使得 Delta2 表的利用率增加。之后减小的原因是，随着 Delta1 表的增大，Delta1 表的平均移位距离变大，在移位时用到 Delta1 表大于 Delta2 表的移位距离，使得 Delta2 表的利用率减少。

同时，Delta2 表的使用概率不高，在 $n=512$ 的情况下，最大的使用概率也不到 1%。并且由解析表达式知，随着 n 的增大（假设保持 m/n 不变），Delta2 的使用概率是逐渐下降的。因此，在 HBM 算法中，性能可以主要靠 Delta1 表中的平均移位距离来分析，这也

是以下内容分析的重要根据。

(2) Delta1 表的平均移位距离。

由分析可知，Delta1 表的平均移位距离是决定算法平均移位距离的主要矛盾。Delta1 表的平均移位距离越大，则算法时间效率越高。同时由于 Delta1 表采用允许碰撞的散列表方式构造，其平均移位距离和表的大小有很大关系，下面将分析 Delta1 表中平均移位距离与 Delta1 表空间大小的关系。

记特征字符串数为 n，长度为 L，块大小为 b，Delta1 表长度为 m。

考虑 Delta1 表中某一项，对于任一长度为 b 的字符块，其散列值落在 Delta1 表中此项的概率是 $p=\dfrac{1}{m}$(假设散列函数的结果均匀分布)，则平均移位距离为

$$
\begin{aligned}
\overline{\text{distance}_{\text{Delta1}}} &= 1\cdot p\cdot 0(1-p)\cdot p\cdot 0+\cdots+(1-p)^{n-1}\cdot p\cdot 0\\
&\quad +(1-p)^{n}\cdot p\cdot 1+(1-p)^{n+1}\cdot p\cdot 1+\cdots+(1-p)^{2n-1}\cdot p\cdot 1\\
&\quad +\cdots\\
&\quad +(1-p)^{(L-b)n}\cdot p\cdot (L-b)+(1-p)^{(L-b)n+1}\cdot p\cdot (L-b)\\
&\quad +\cdots+(1-p)^{(L-b+1)n-1}\cdot p\cdot (L-b)\\
&\quad +(1-p)^{(L-b+1)n}(L-b+1)\\
&=(1-p)^{(L-b+1)n}(L-b+1)+\sum_{i=0}^{L-b}\sum_{j=0}^{n-1}(1-p)^{in+j}\cdot p\cdot i
\end{aligned}
\tag{5-12}
$$

式(5-12)右边的多项式第一项，表示第一个特征字符串的最后长度为 b 的字符块，如果散列值落在 Delta1 表中此项，则此项存放 0 值。之后的项含义类似，分别考察接下来的每个特征字符串的最后一个字符块，所以第一行遍历了所有特征字符串的最后长度为 b 的字符块，也即得到了 Delta1 表中此项为 0 的概率。同理，计算 Delta1 表中此项为 1，2 直至 $L-b$ 的情况。多项式的最后一项是所有特征字符串的字符块都不出现的情况，此时 Delta1 表中此项为 $L-b+1$。

记 $a=(1-p)^n=\left(1-\dfrac{1}{m}\right)^n$，化简式(5-12)得到式(5-13)：

$$
\begin{aligned}
\overline{\text{distance}_{\text{Delta1}}} &= a^{(L-b+1)}\cdot (L-b+1)+(1-a)\sum_{i=0}^{L-b}a^{i}\cdot i\\
&=\frac{a-a^{l-b+2}}{1-a}
\end{aligned}
\tag{5-13}
$$

在 m 较大的情况下，a 可以近似化简为 $a=\left(1-\dfrac{1}{m}\right)^n=\left(1-\dfrac{1}{m}\right)^{-m\cdot\frac{n}{-m}}\approx \mathrm{e}^{-\frac{n}{m}}$，则此时，

$$
\overline{\text{distance}_{\text{Delta1}}}=\frac{\mathrm{e}^{-\frac{n}{m}}-\mathrm{e}^{-\frac{n}{m}(l-b+2)}}{1-\mathrm{e}^{-\frac{n}{m}}}
\tag{5-14}
$$

由式(5-14)可知，当 Delta1 表较大时(即 m 比较大)，为了达到相同的平均移位距离，m 与 n 基本成正比例关系，因此，只需要考虑平均移位距离和两者比值的关系。记 x 为 Delta1 表大小 m 与特征字符个数 n 的比值。Delta1 表平均移位距离 $\overline{\text{distance}_{\text{Delta1}}}$ 与 x 的关系如图 5.20 所示。

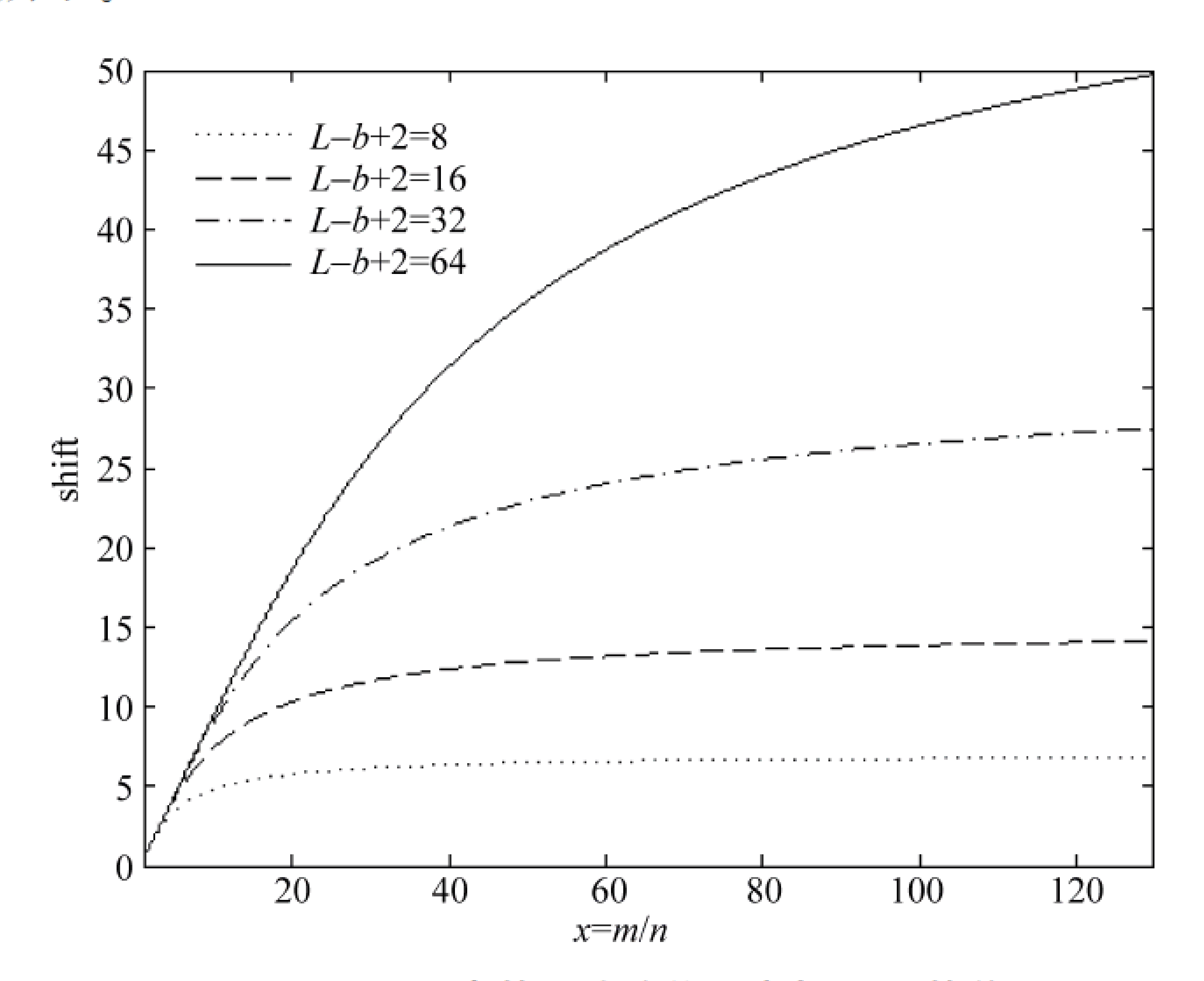


图 5.20　Delta1 表的平均移位距离与 m/n 的关系

直观上来说，就是随着特征字符串长度的增加，Delta1 表需要的空间也需要逐渐增加，从而能够减少散列碰撞的概率，增加平均移位距离，进而提高算法的时间效率。

同时由图 5.20 可知，随着两者比值的增大，平均移位距离也随之增加，但增长速度逐渐减缓。对于 $L-b+2$ 较小的下面 3 条曲线来说，当 $m/n>40$ 以后，随着空间增大，Delta1 表的平均移位距离的增长微乎其微。也就是说，在这种情况下，只需要消耗较小空间存放 Delta1 表就能达到相近的效率。

(3) Delta1 表的大小的选择。

回顾本章提到的几种多模匹配算法，虽然同样采用了 BM 算法的思想，但是由于不使用散列方式(前面的解析分析也同样适合)，m 的大小只可能有固定的几种取值，哪怕仅使用最小的 2 个字符的字符块作为比较单元，Delta1 表的大小也将达到 $m=65\,536$ 项，如果此时特征字符串数目为 $n=512$，那么 $m/n=128$，由图 5.20 可知，这种方式的空间效率是非常低的。

由微积分可知，此算法的时间空间效率平衡的最佳的点在曲线的拐点(2 阶导数为 0)。也就是在此点之前，随着空间的增加，时间效率的增加越来越快；而在此点之后，随着空间的增加，时间效率的增加越来越慢，乃至趋近于 0。而这种减缓得趋势随着 $L-b+2$

的增加而减小。

此曲线的拐点可以通过数值方式解得，下面给出了拐点处的 x 值(即 m/n)与 $L-b+2$ 之间的关系图。由图 5.21 可知，这个时空效率平衡的最佳点，随着 $L-b+2$ 的增加，基本上成线性增长。

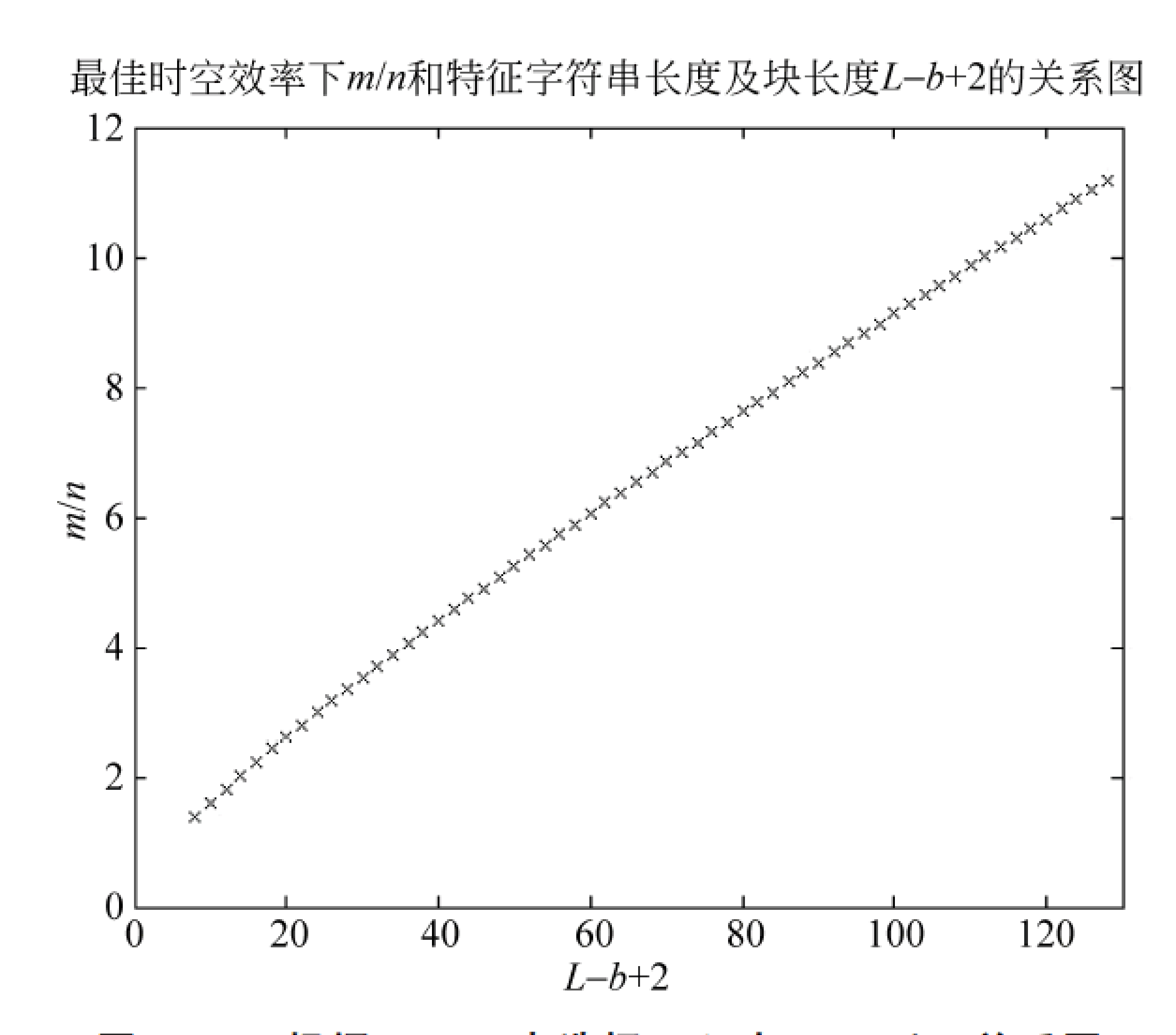


图 5.21　根据 Delta1 表选择 m/n 与 $L-b+2$ 关系图

(4) 算法的平均移位距离。

算法的平均移位距离 $\overline{\text{distance}_{\text{comp}}}$ 可以通过表 5.5 计算。

表 5.5　移位发生时的平均移位距离表

j	Possibility of skip happening	Skip distance
$j=0$	$P(j=0)=a$	skip_0
$j=1$	$P(j=1)=(1-a)\cdot a^2$	skip_1
$j=2$	$P(j=2)=(1-a)\cdot(1-a^2)\cdot a^3$	skip_2
…	…	…
$j=k$	$P(j=k)=(1-a)\cdot(1-a^2)\cdots(1-a^k)\cdot a^{k+1}$	skip_k
match	$P(\text{match})=(1-a)\cdot(1-a^2)\cdots(1-a^{k+1})$	1

其中，j 在移位发生前已经成功比较的次数 $k=L-b$，$P(j=i)$是此时发生移位的概率，$\text{skip}_i(i=0,1,\cdots,k)$是此时的平均移位距离。当仅仅考虑 Delta1 表而忽略 Delta2 表时，平均移位距离 $\overline{\text{distance}_{\text{comp}}}$(见图 5.22)可以近似计算得到式(5-15)：

$$\overline{\text{distance}_{\text{comp}}} \approx \sum_{i=0}^{k} \frac{\text{skip}_i \cdot P(j=i)}{i+1} + 1 \cdot P(\text{match})$$

$$= \sum_{i=0}^{k} \frac{(1-a^{L-b+2-i}) \cdot a^{i+1} \cdot \prod_{t=0}^{i}(1-a)^t}{(1-a)\cdot(i+1)} + \prod_{t=0}^{k+1}(1-a)^t \qquad (5\text{-}15)$$

图 5.22 算法平均移位距离与 m/n 关系

同样，用类似上一小节的方法，可以根据平均移位距离，来选择性价比最高的情况下的参数 m/n 的近似值，如图 5.23 所示。

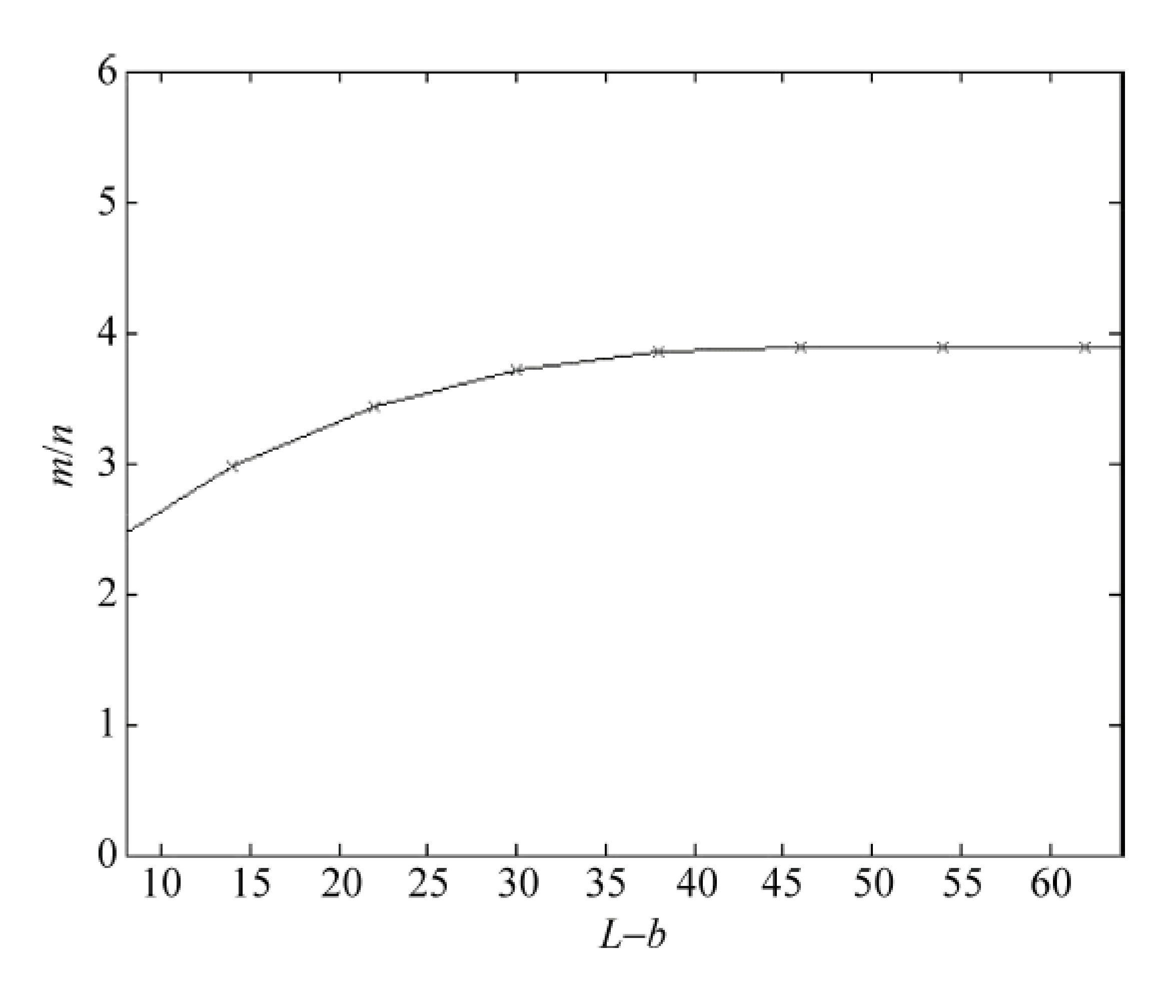


图 5.23 根据平均移位距离表选择 m/n 与 $L-b+2$ 关系图

图 5.24 给出了 HBM 算法和 WM、AC 算法的平均移位距离比较。就性能来说，WM 算法的最佳情况和 HBM 算法相差不多，但是 WM 算法无法动态改变启发式移位表的空间大小，所以其平均移位距离是成阶梯形增长的，即每次跃变发生在字符块大小的改变（如字符块单位由 2B 变化为 3B 时），因此不像 HBM 算法那么灵活。AC 算法采用的匹配思路不同，因此，其平均移位距离总是 1。

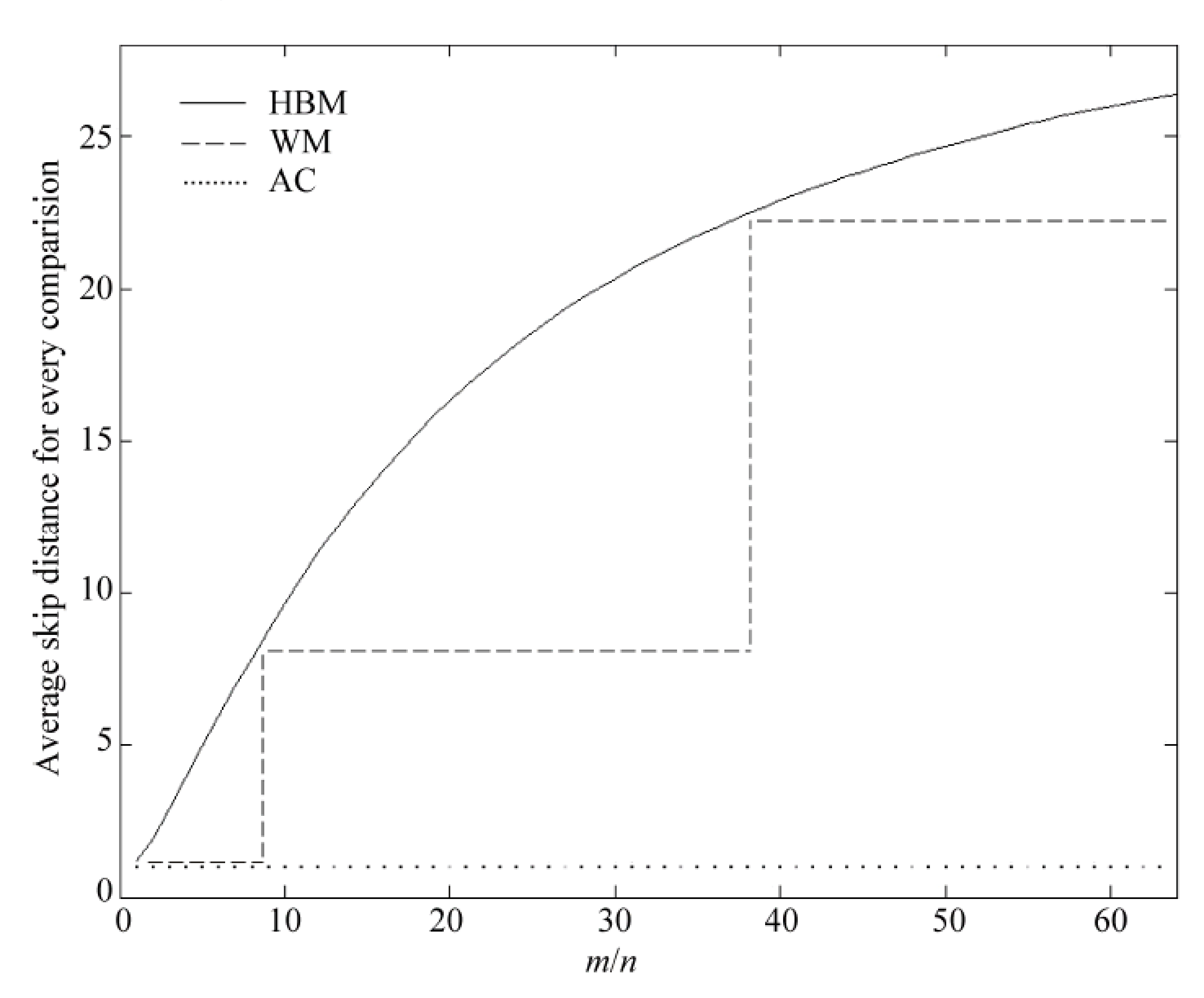


图 5.24　HBM、WM 和 AC 算法的平均移位距离比较

2）误判概率分析

由于在多模匹配的算法下，当一次比较完全“成功匹配”后，还需要进行精确匹配。如果精确匹配发现不符合任何一个特征字符串，则称此次匹配为“伪匹配”，也称为误判。

不论是否使用散列方法，这种精确匹配都存在。因此，计算需要进行这种精确匹配的概率显得必要。对于任意随机的二进制字节流，某次比较出现误判的概率为式(5-16)：

$$P_{\text{false-positive}} = \left[1-\left(1-\frac{1}{m}\right)^{n}\right]\cdot\left[1-\left(1-\frac{1}{m}\right)^{2n}\right]\cdots\left[1-\left(1-\frac{1}{m}\right)^{(L-b+1)n}\right]-\frac{n}{2^{L}}$$
$$=\prod_{j=1}^{L-b+1}\left[1-\left(1-\frac{1}{m}\right)^{jn}\right]-\frac{n}{2^{L}} \tag{5-16}$$

上式连乘的第 j 个因子表示第 j 次的字符块比较成功，后面减去的项表示精确比较成功（即此次是个真正的特征字符串）。

在一般情况下，$\frac{n}{2^L}$项可以忽略；同时，在 m 较大的情况下，$\left(1-\frac{1}{m}\right)^{n}=\left(1-\frac{1}{m}\right)^{-m\cdot\frac{n}{-m}}\approx e^{-\frac{n}{m}}$。出现误判的概率可以近似为式(5-17)：

$$P_{\text{false}-\text{positive}} \approx \prod_{j=1}^{L-b+1} [1 - e^{-\frac{jn}{m}}] \tag{5-17}$$

记 x 为 Delta1 表大小 m 与特征字符个数 n 的比值。则出现误判概率 $P_{\text{false}-\text{positive}}$ 与 x 的关系如图 5.25 所示。

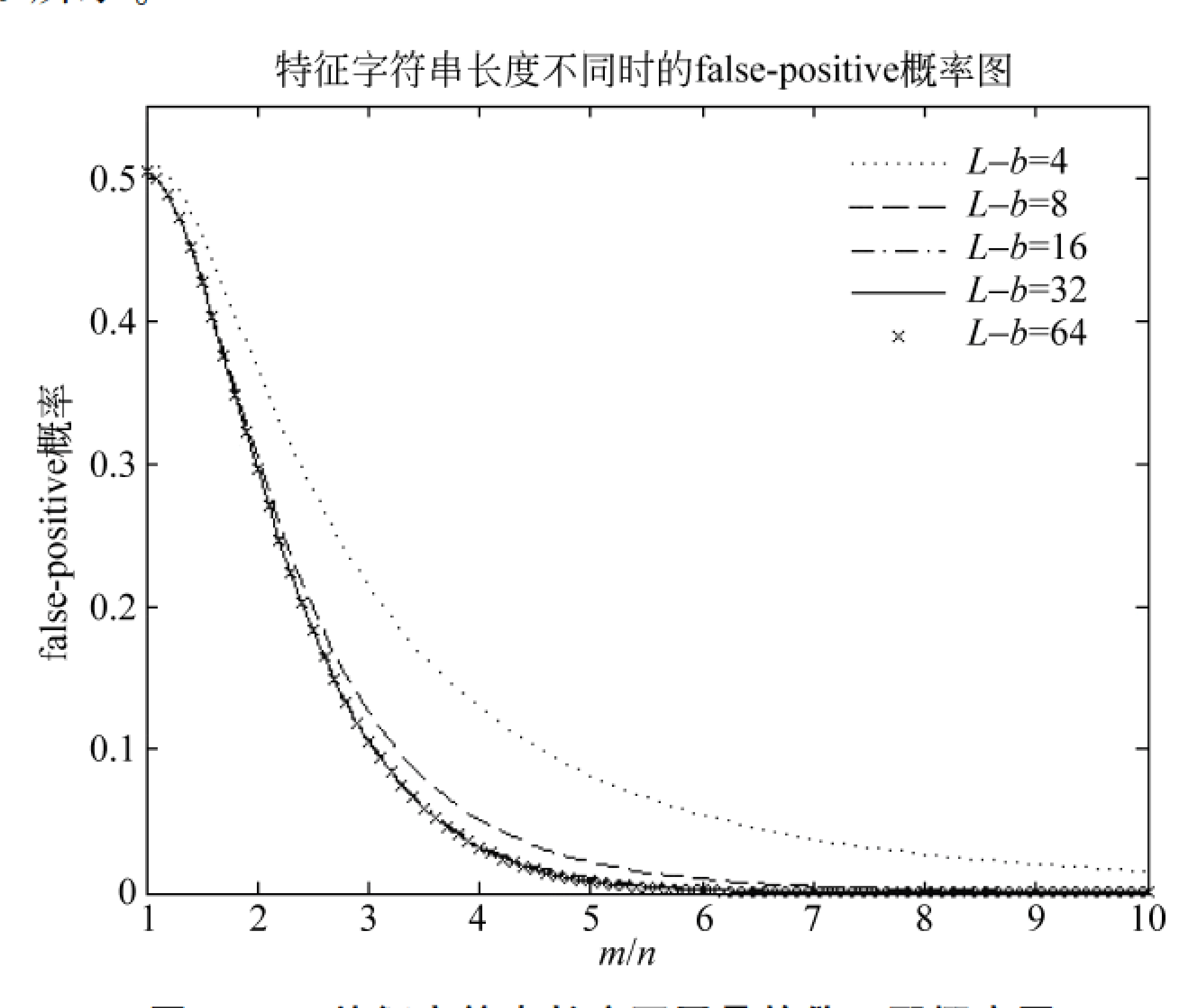


图 5.25 特征字符串长度不同是的伪匹配概率图

由图 5.25 可知，当字符串长度达到一定长度后，false-positive 概率基本相同，即主要取决于连乘式的前几项。

图 5.26 比较了 HBM 算法和 WM、RSI 算法的误判概率情况。WM、RSI 算法都是仅

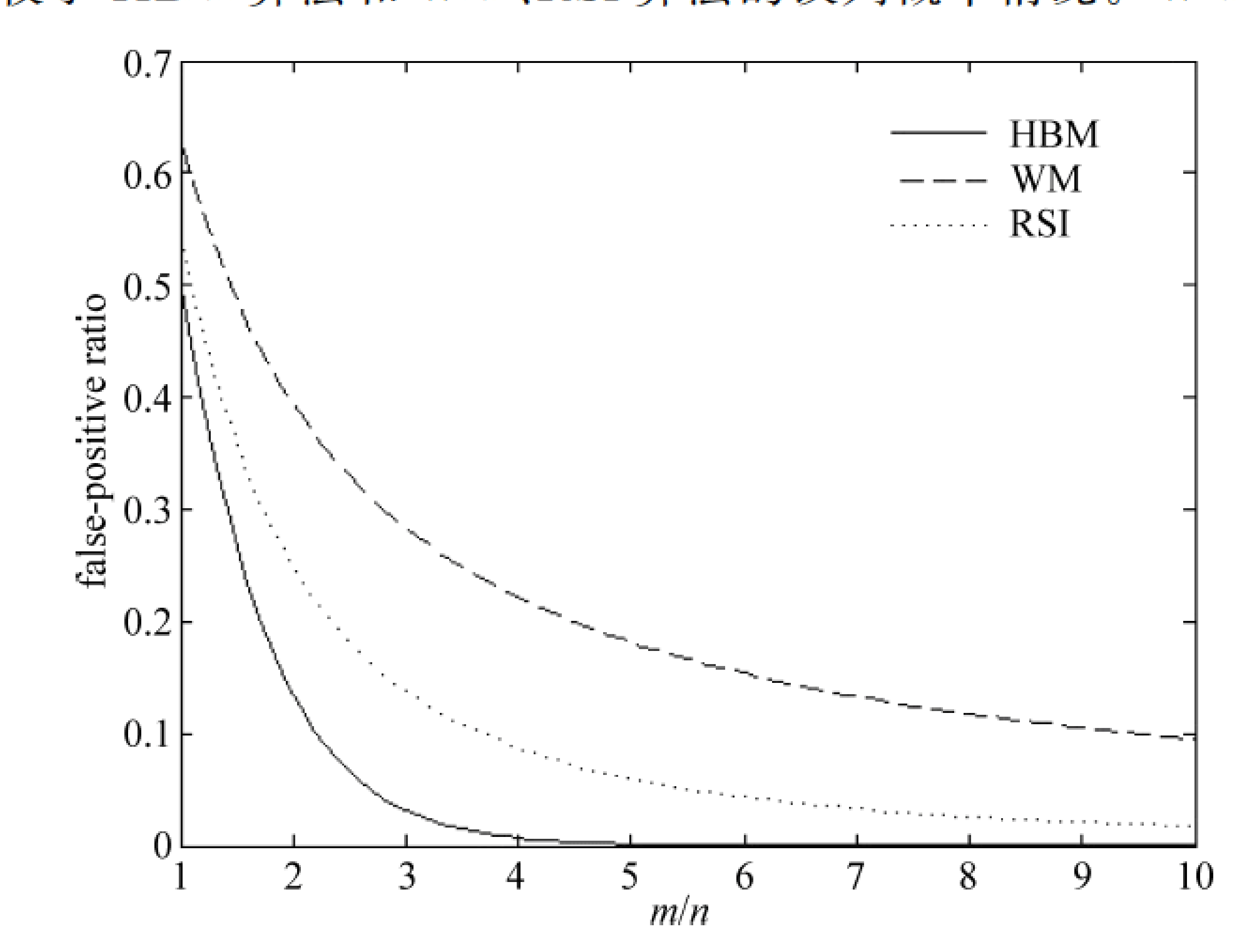


图 5.26 HBM 算法和 WM、RSI 算法的伪匹配概率比较图

仅使用最后一个或几个字符块来构在 Delta1 表。因此，使用 HBM 算法在整体的误判概率上是有很大优势的，减少了精确比较的次数，提高了算法效率。AC 算法由于没有误判情况，因此未进行比较。

5.5　实验与分析

5.5.1　实验环境

在测试床中对系统的有效性进行了测试。该测试床包括以下设备：千兆以太网络交换机(TP-LINK TL-SL2226P+，24+2G)、流量发生器 IXIA 1600，ENP 2611 实验板(含 IXP2400 Network Processor)，以及携带特征码的 PC、多台普通 PC 和服务器。其测试环境示意图如图 5.27 所示。

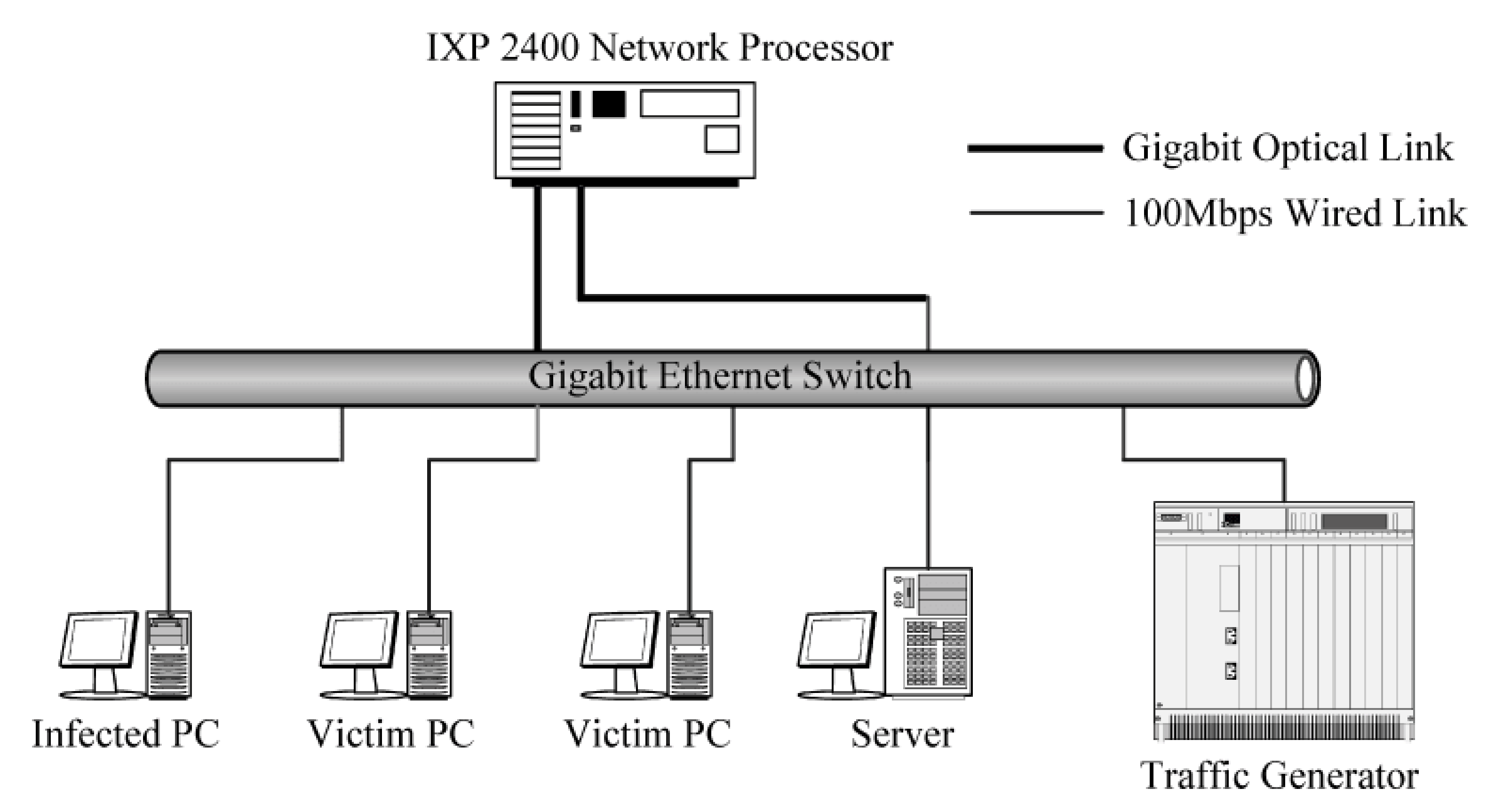


图 5.27　测试环境示意图

反特征码系统可以工作在监听模式和转发模式。前者指被动接收网包，而后者指主动接收、处理和转发网包。这里只进行了监听模式的测试。

实验中分别用流量发生器产生带有特征码特征的 IP 网包，以及用蠕虫感染主机发出带有病毒代码的 IP 网包，该系统均能正确检测并报警。

5.5.2　基于 Bloom Filter 算法的引擎性能

文本在 Intel 通用多核平台上实现了基于 Bloom Filter 算法的匹配引擎，并进行了性能测试实验。在实验中，蠕虫特征码和 Bloom Filter 的算法参数设置如下。

位向量大小：$m=2048$。

蠕虫特征码库大小：$n=128$。

散列函数个数：$k=8$。

Intel 通用多核平台的 XSCAL 核的工作主频为 400MHz，每个微引擎(ME)的工作主频为 400MHz，SRAM 和 DRAM 的总线主频为 100MHz，其吞吐率值均取于媒体总线(Media Bus)上，其时钟频率为 104MHz。

输入的网络网包流，载荷长度大小分为 64、128、256、512 和 1024B。其中每组网包中携带病毒特征码的比例(称为命中率)又分别为 0%、20%、40%、60%、80%和 100%。系统吞吐量定义为系统输入为满负荷时的最大的输出速率。由于系统需要逐一字节扫描整个网包载荷，因此系统吞吐量、网包载荷大小和携带特征码的比例紧密相关。当网包的载荷长度增加时，吞吐量将会下降。图 5.28 给出了系统吞吐量的测量值。

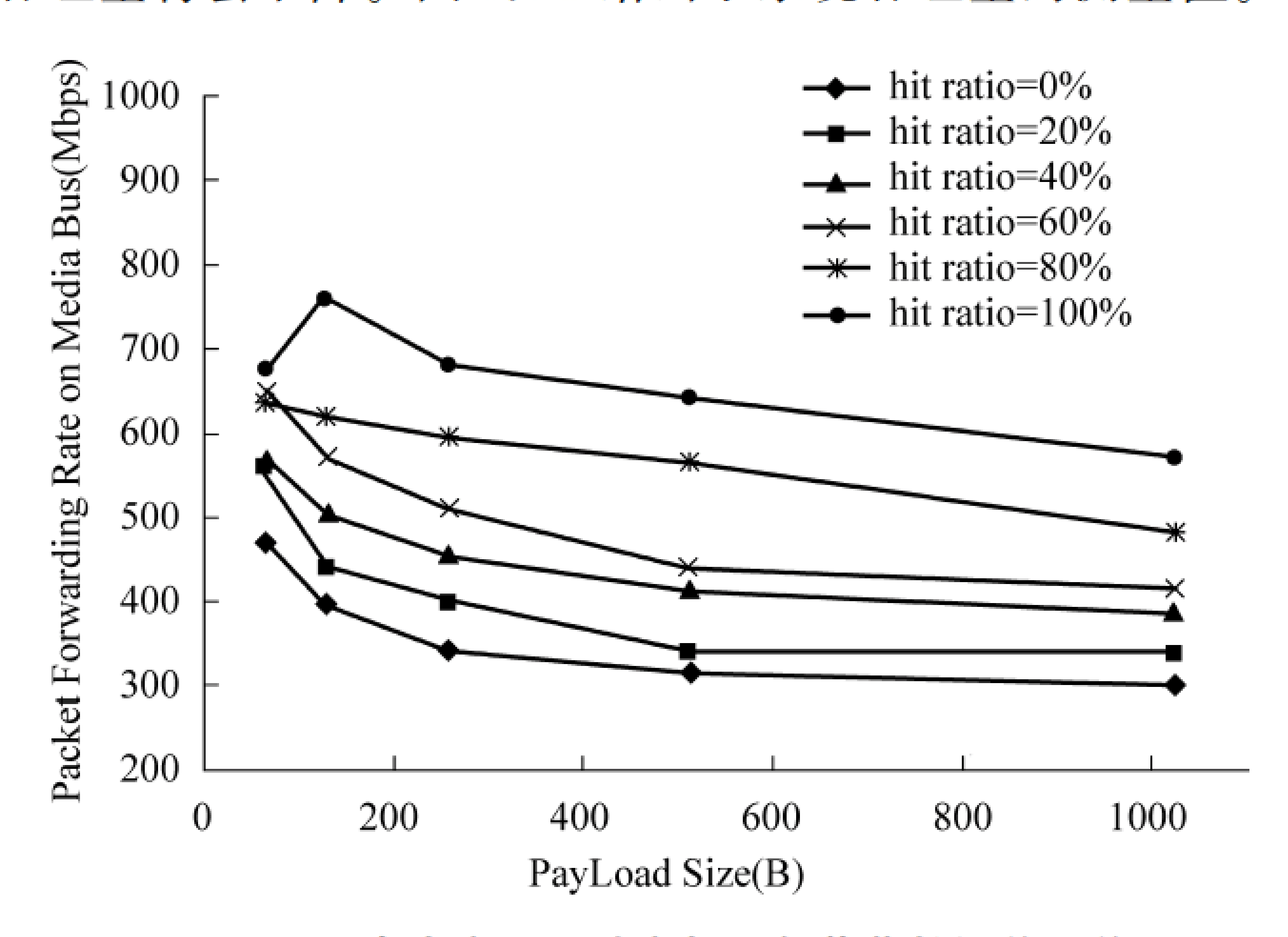


图 5.28 不同命中率下吞吐率与网包载荷长短关系关系图

系统时延以时钟周期(Media Bus)为单位来测量。实验记录了处理不同载荷长度的网包时延，这些网包不携带病毒特征码，所以其处理时延是最坏情况。图 5.29 给出了系统时延的测量值和网包长度的关系，可以看到基本呈线性关系。

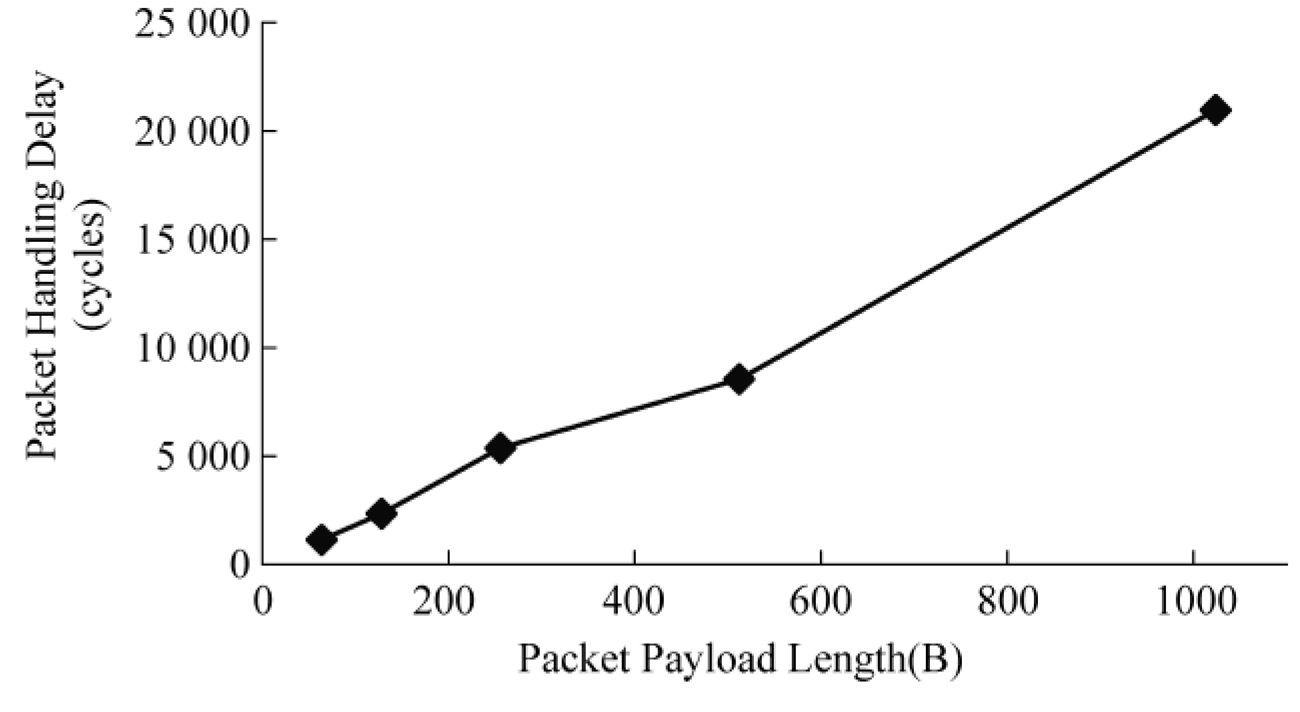


图 5.29 网包处理的系统时延和网包长度的关系

由实验数据可以看到，基于 Bloom Filter 的匹配引擎，由于不太适合通用多核平台体系结构，因而其吞吐率性能指标并不十分理想，未能达到 1Gbps 的速度。

5.5.3　基于 HBM 算法的引擎性能

本章在 Intel IXP 多核平台上实现了 HBM 算法，并对性能进行了实验。实验采用随机产生的固定长度特征码字，字长为 32B，特征码字数目为 512 个（注意：HBM 算法也支持非固定长度的特征码字）。产生的以太网帧长度分别为 118，246，…，1398，以及最大可能的 1518B。网包中携带特征码字的比例（简称为命中率）为 0%，20%，…，100%，特征码字在网包载荷中的出现位置随机决定。实验输入的网包数目为 4000。

Intel IXP 多核平台的 XSCAL 核工作主频为 400MHz，每个微引擎（ME）的工作主频为 400MHz，SRAM 和 DRAM 的总线主频为 100MHz，其吞吐率值均取于媒体总线（Media Bus）上，其时钟频率为 104MHz。

在系统结构上，控制平面的 XSCAL 核载荷初始化 Delta1 表和 Delta2 表，同时在 DRAM 中建立散列链表方式组织的特征字符串集合；数据平面采用了 6 个微引擎，其中 1 个微引擎用于网包接收模块，1 个微引擎用于网包发送模块，其余 4 个微引擎运行 HBM 算法过滤网包载荷。

在运行 HBM 的微引擎上，Delta1 表存放在 Local Memory 中，每个表项需要 5b 来表示移位距离，则一个长字（32b）能存放 6 个表项，所以 Delta1 表的最终大小为 640×5＝3.2K 项。Delta2 表存放在邻居寄存器（Next Neighbor）中，采用本地微引擎存取模式（Self 模式）。需要匹配的特征码字以散列链表的方式存放在 DRAM 中。

图 5.30 和图 5.31 反映了匹配引擎的吞吐率性能。图 5.30(a)是命中率为 0%时的吞吐率随以太帧长度变化情况，即没有网络网包包含蠕虫特征码的情况。当帧长度很短时，时间开销主要集中在帧头和包头的分类处理上，深度包检测的开销不大，因而吞吐率

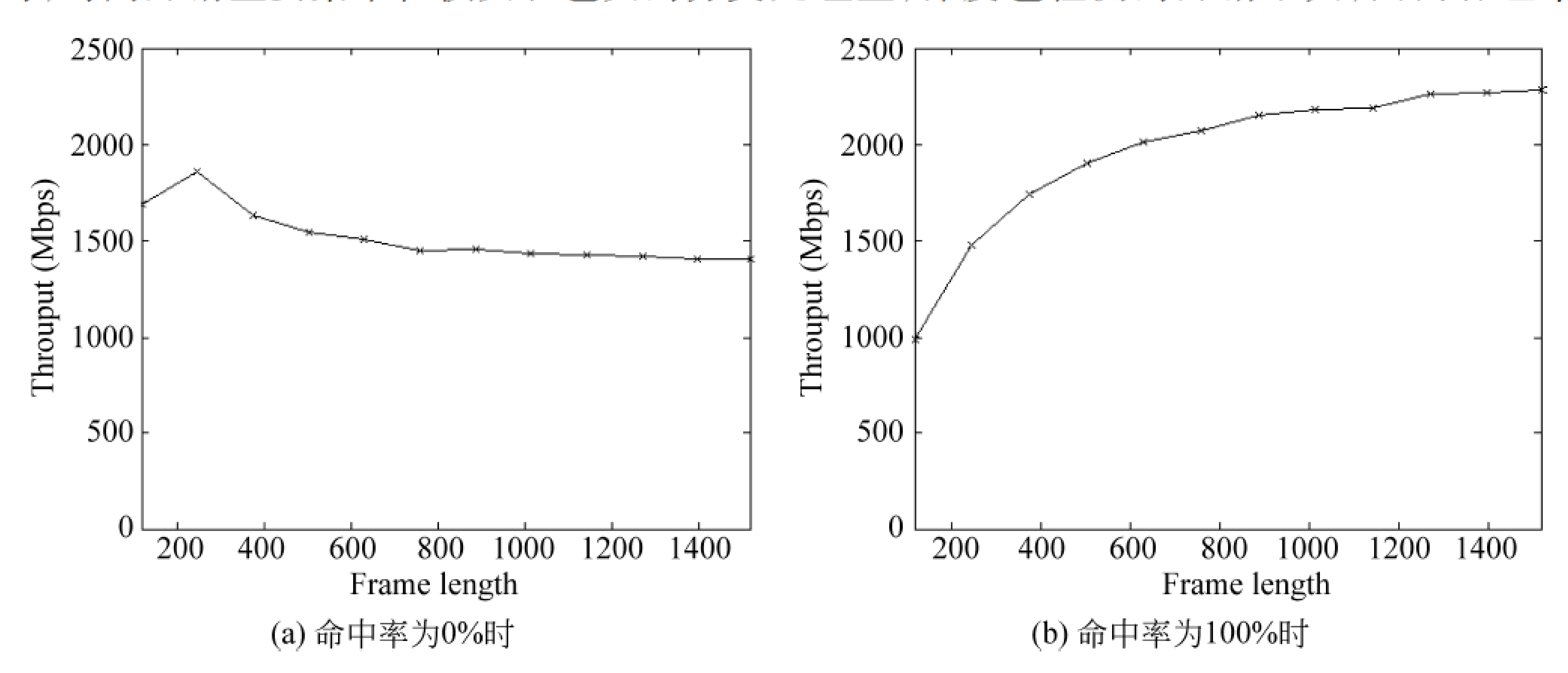


图 5.30　吞吐率随帧长变化情况

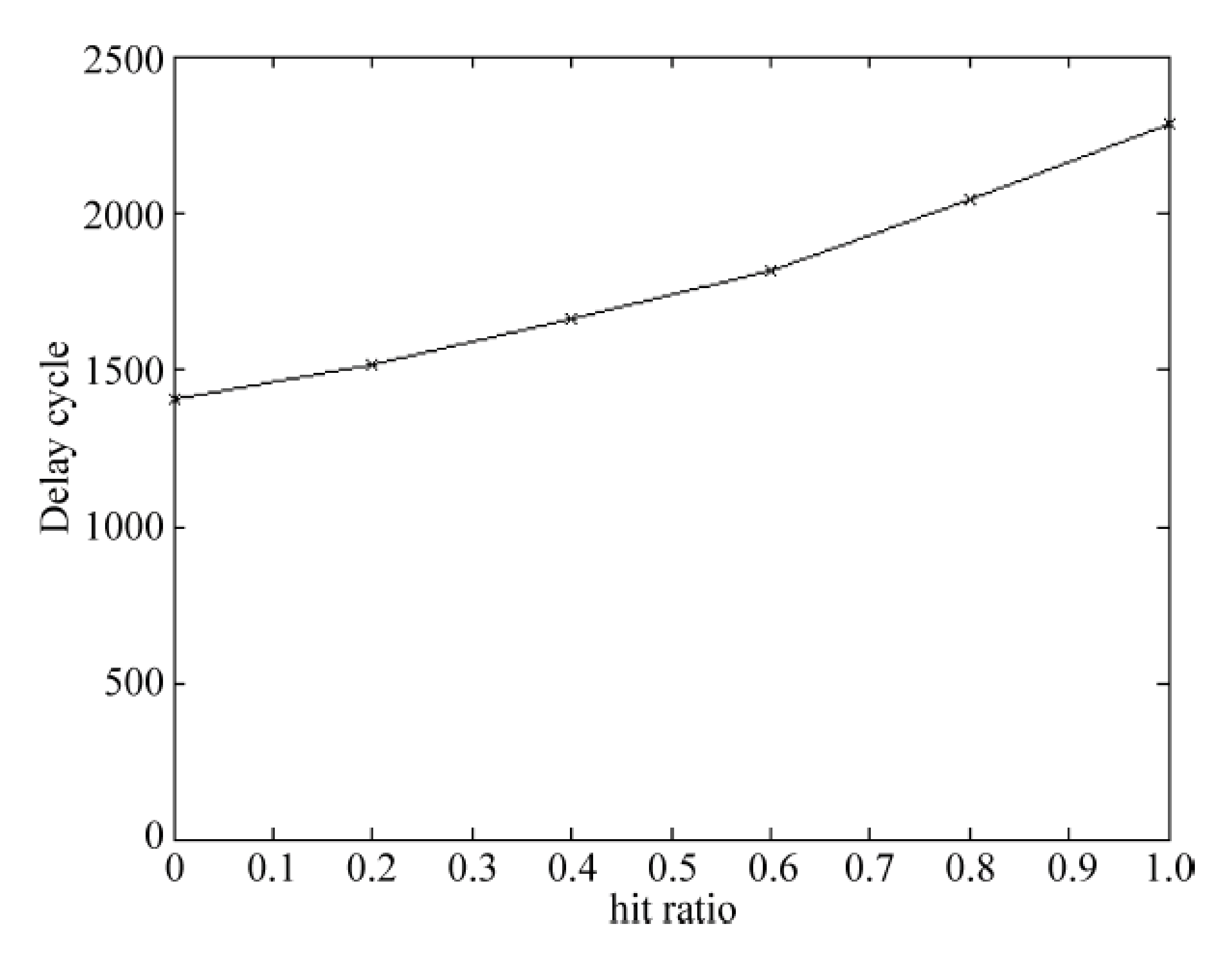


图 5.31 吞吐率随命中率的变化情况(帧长固定为 1518B)

偏高;当帧长度变长时,深度包检测逐渐成为了主要性能瓶颈,吞吐率缓慢下降,最后稳定在 1400Gbps 到 1500Gbps 之间。图 5.30(b)是命中率为 100%时的情况,即每个网络网包都包含蠕虫特征码的极端情况。此时,每帧的处理都会有特征码精确比较的额外时间开销,但同时一旦发现特征码即结束操作,剩余的载荷内容工作不必继续检测。因而,帧长越长,可以不必检测的载荷越多,吞吐率也随着以太帧长度的增加而增加。在图 5.31 表示的吞吐率随命中率变化情况中,也同样可以看到这种趋势,即随着命中率增加,每帧平均不需要检测的载荷数目增多,吞吐率增大。图 5.32 反映的不包含蠕虫特征码的网络网包通过匹配引擎的延时情况。这里的时延周期取自通用多核平台的介质总线上,其时

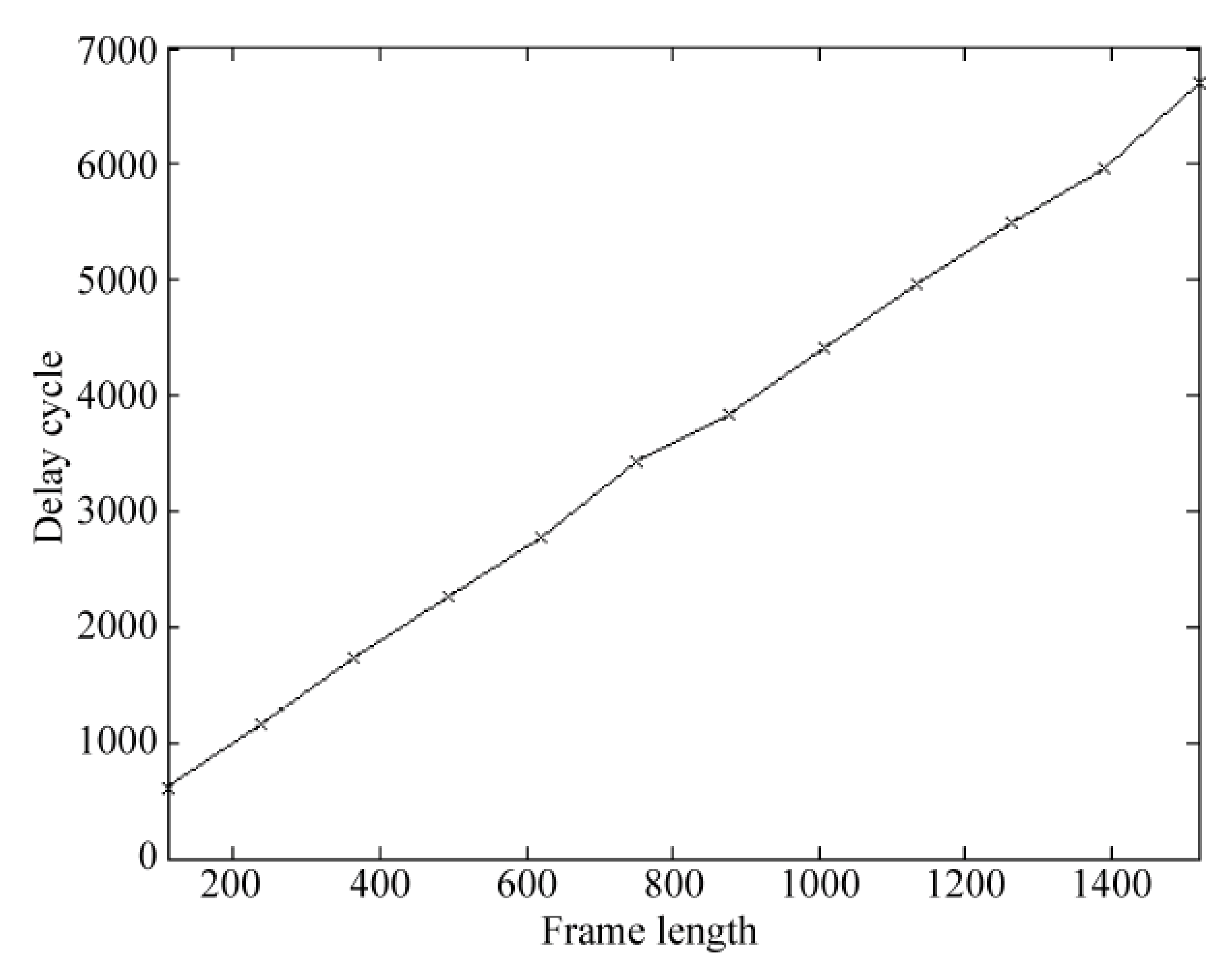


图 5.32 单帧通过匹配引擎的延时随帧长的变化情况(命中率为 0%)

钟频率为 104MHz。根据图中数据，随着帧长度的增加，单帧处理的平均延时呈线性平稳增长。

由以上性能数据知：在 IXP2400 上采用 HBM 算法的匹配引擎，已经达到了千兆以太网的吞吐率要求；同时，算法的吞吐率性能稳定，在蠕虫爆发时(即命中率大幅度提高)，性能不会下降很多。

5.6　本章小结和展望

本章的主要研究内容是基于已有的特征码来匹配和检测攻击流量，定位和发现攻击流量。多核平台由于具有灵活的可编程特性，高性能和适用于高速的网包处理的能力而备受关注。本章主要介绍了采用 Intel IXP 多核平台，设计和实现基于深度包检测的反病毒引擎。

同时，针对 TCP/IP 流状态内容扫描和跨 TCP 包的蠕虫检测，本匹配引擎进行了功能的有效扩展。重点设计了流状态的记录格式，实现流状态的检索和维护，以及跨包的蠕虫特征码的检测。

本匹配引擎系统的主要特色是采用并行 Bloom Filter 算法和 HBM 算法，从而达到了较好性能指标。特别是基于 HBM 算法的引擎，系统性能稳定，且达到了千兆以太网的要求。

HBM 算法是对经典 BM 算法的一个推广，其核心想法是以字符块为比较单位和以散列运算作为比较操作。算法的重点是需要重新设计 Delta1 表和 Delta2 表。在该算法的实现上，根据 NP 的存储器架构存放相应的数据结构，减少了 I/O 冲突。实验仿真表明 HBM 算法有效地提高了吞吐率，并且性能指标稳定，达到了千兆以太网标准，符合高速网络环境下的性能要求。

网络攻击的检测、发现和阻止是一个比较复杂的问题，往往需要 Internet 紧急响应组，计算机操作系统和应用软件商，计算机安全产品商，网关、防火墙和路由器设备制造商分工协作完成。

对于新爆发的攻击往往无能为力，至于病毒特征码的产生则需要参考蜜罐系统的捕获，计算机安全产品商发布的病毒特征来获取。而发现病毒流量后采取的动作则需要参考网关、防火墙和路由器设备制造商的防御措施。

未来的进一步工作体现在扩大特征码的特征码数据库、减小误判概率，以及提高反特征码系统的吞吐率性能和时延特性，使之成为现实可用的系统。

参 考 文 献

[1] Jia Ni, Chuang Lin, Zhen Chen et al., A fast multi-pattern matching algorithm for deep packet inspection on a network processor, International Conference on Parallel Processing (ICPP), 2007.

[2] Zhen Chen, Chuang Lin, and Jia Ni et al., AntiWorm NPU-based Parallel Bloom filters for TCP/IP Content Processing in Giga-Ethernet LAN. The First IEEE LCN Workshop on Network Security, 2005.

[3] Zhen Chen, Chuang Lin, and Jia Ni et al., AntiWorm NPU-based Parallel Bloom filters in Giga-Ethernet LAN, IEEE International Conference on Communications (ICC), 2006.

[4] 林闯,郑波,倪嘉,陈震. 基于通用多核平台的高速多维报文分类算法的设计和实现. 中国, 200510011854,[p]. 2005-6-3.

[5] 倪嘉. 基于通用多核平台的高速匹配引擎设计与实现(指导教师:林闯). 清华大学硕士毕业论文,2007 年.

[6] Snort. http://www.snort.org.

[7] Bro. http://www.bro-ids.org.

[8] Suricata. http://www.suricata-ids.org.

[9] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422~426.

[10] Alfred V. Aho, and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. Communications of the ACM 18.6 (1975): 333~340.

[11] Robert S. Boyer, and J. Strother Moore. A fast string searching algorithm. Communications of the ACM 20.10 (1977): 762~772.

[12] Sun Wu, and Udi Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, Vol. 84. University of Arizona, 1994.

[13] Mike Fisk, and George Varghese. An analysis of fast string matching applied to content-based forwarding and intrusion detection. Technical Report CS2001-0670. University of California in San Diego, 2002.

[14] C. Jason Coit, Stuart Staniford, and Joseph McAlerney. Towards faster pattern matching for intrusion detection, or exceeding the speed of snort. In proceeding of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX Ⅱ),2002.

[15] Bo Xu, Xin Zhou, and Jun Li. Recursive Shift Indexing: A Fast Multi-Pattern String Matching Algorithm. In proceeding of the 4th International Conference on Applied Cryptography and Network Security (ACNS 2006), 2006.

[16] Zongwei Zhou, Yibo Xue, Junda Liu, Wei Zhang, and Jun Li. MDH: A High Speed Multi-phase Dynamic Hash String Matching Algorithm for Large-Scale Pattern Set. Information and Communications Security, 2007.

[17] Rong-Tai Liu, et al. A fast pattern-match engine for network processor-based network intrusion detection system, Information Technology: Coding and Computing. Information Technology: Coding and Computing. International Conference Vol. 1, Page(s):97～101, 2004.

[18] Donald E. Knuth, James H. Morris Jr, and Vaughan R. Fast Pattern Matching in Strings. SIAM journal on computing 1977, 6(2): 323～350.

[19] Sarang Dharmapurikar, Praveen Krishnamurthy, T. S. Sproull, J. W. Lockwood. Deep packet inspection using parallel bloom filters. High Performance Interconnects, 2003.

[20] ClamAV. http://www.clamav.net.

[21] RFC1321. http://www.ietf.org.

第 6 章

互联网流量攻击检测实例

6.1 数据中心的服务监测

互联网服务一般部署于数据中心(Data Center),一个典型的数据中心架构如图 6.1 所示。最前端是 1+1 冗余接入路由器,然后是核心路由器、核心交换机、负载均衡器,最后是接入交换机。这些网络设备连接着 Web 服务器、Web 缓存服务器、数据库服务器等。

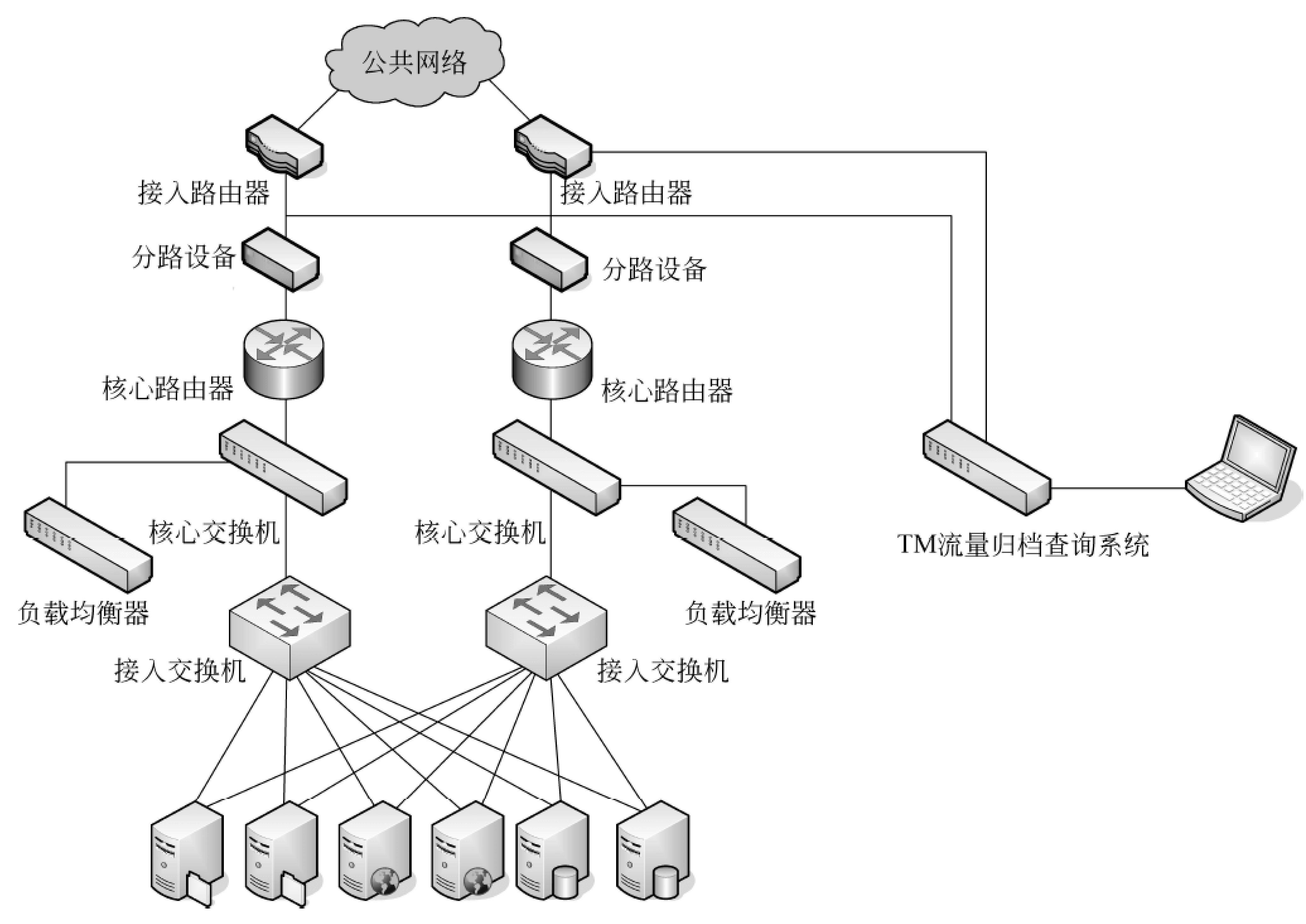


图 6.1 数据中心出口的服务监测

旁路网络攻击检测一般在互联网数据中心出口处，部署了网流信息归档与查询系统。采用分路设备，将流量镜像出来，对流量进行旁路后，进行流记录的存储，或者对流量内容进行存储，再进行深度包检测，分析检测可能的攻击特征，如图 6.1 所示。

6.2　互联网服务访问行为分析

互联网数据中心出口部署了网流信息归档与查询系统，该系统在网络上监控万兆级数据中心发生的每一个请求、服务、交易；对途经流量进行深层次报文分析(DPI)，重组应用层有效信息，双向匹配请求/服务并记录。同时，在线实时计算每一个服务请求的响应时间，为大数据分析做准备。

统计用户 Web 访问的元信息，包括源 IP 地址、域(domain)和资源标识(URI)。统计会话级(Session 级)的访问特征，合并用户 Web 访问到会话级(Session)，统计会话内的访问特征：访问次数、访问深度、访问宽度、Agent 个数、Get 文件访问比例、静态文件访问比和请求比等等。

通过以上特征，机器学习判断是何种行为：是正常用户行为、爬虫行为、黑客扫描行为，还是攻击行为，制定相应的安全策略。分析用户的行为，如正常访问(login、browsing、order、payment)的行为；异常的攻击行为，如非对称 DoS 攻击(Asymmetric DOS Attacks) 和 HashDOS 攻击(通过造成散列表的长串 pipeline 来消耗计算资源，加长响应时间等)。

例如，可以通过统计单位时间内某 IP 访问某些热点线路(URL)的次数过大，特别是刷票软件；防止爬虫(Web Crawler)抓取，保护商业情报，比如单位时间内某 IP 访问过的 URL 的次数过大，1s 内大于 30 次的 URL 访问。也可以对 DDoS 攻击的预警，如单位时间里若干个 IP 对某 URL 的访问次数过大，30s 内上百次。

统计攻击行为，对攻击位置与频度的统计，使用黑名单屏蔽掉可能的攻击地址。另外网流归档查询系统也可以查询用户延时信息，发现服务异常，诊断存在的性能瓶颈问题：①服务器响应时间；②应用层上响应时间；③TCP 重组问题还是 HTTP 问题等。

应用开源软件实现方案如图 6.2 所示。为了能够进行高速统计，只抓取了 NetFlow-v9 格式数据，有效统计数据直接存放在内存中。为了避免索引空间消耗过大，采用了压缩方法，具体参见第 3 章内容。

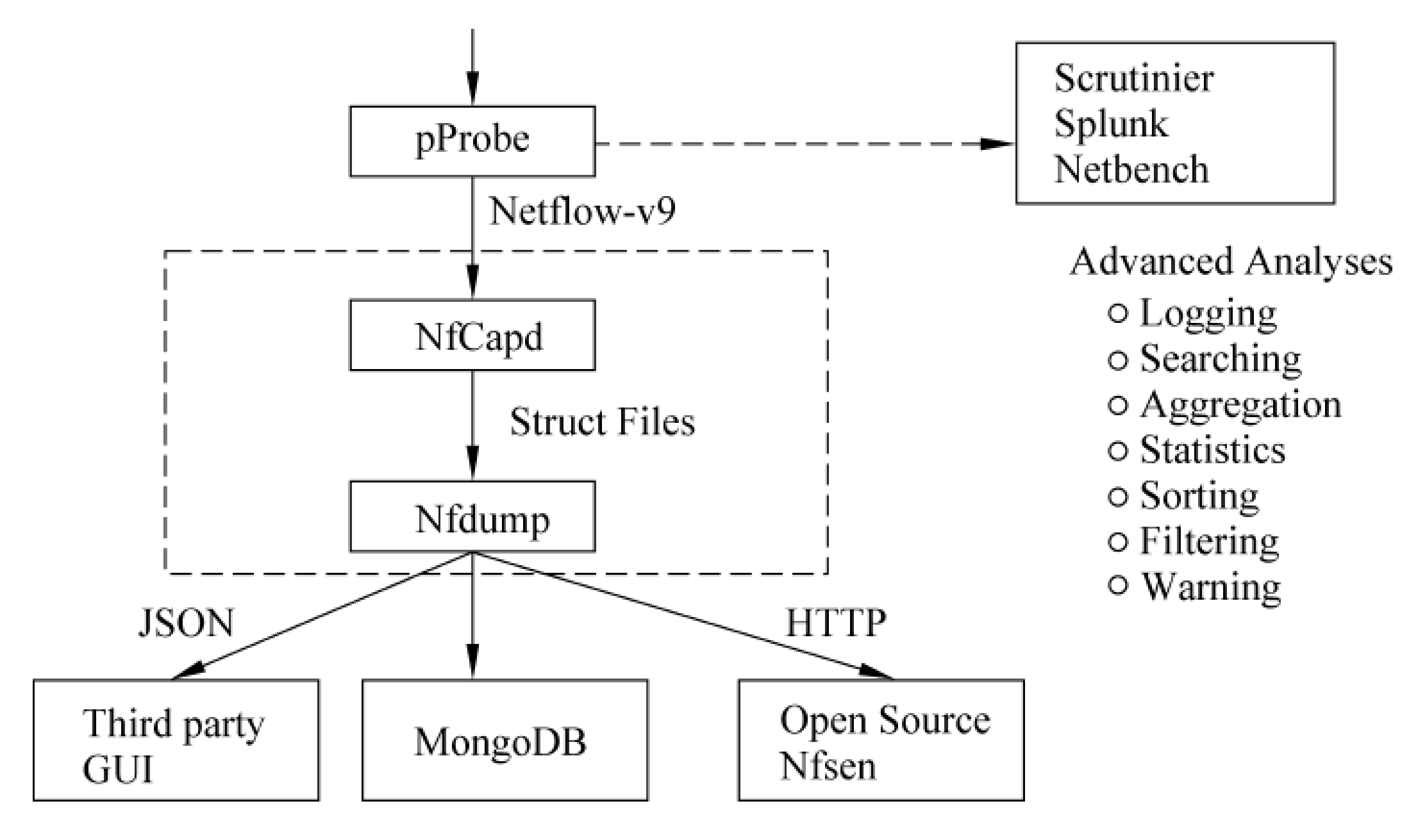


图 6.2 网包获取与离线分析

6.3 互联网服务抗 DDoS 攻击

百度公司拥有世界上最大的中文搜索引擎,每天用户访问几十亿次。百度通过大规模流量分析来进行网络攻击检测,进行 DDoS 防御。通过自主研发的流量镜像系统,获得流量,分析流量,对安全攻击报警。

主要技术包括通过光纤分光器,在不影响业务的情况下获得数据中心的所有流量。如果有多个数据中心的话,可以进行归集处理。通过万兆交换机的光纤镜像接口,用 Trunk 技术,将流量 1∶N 方式物理分流至数据接收分布式集群。分布式集群采用统计 Web 访问数据,存储记录流量,然后通过数据分析集群,汇总统计信息,深入流量数据挖掘。具体部署如图 6.3 所示。

网络流量深入分析还可以评估网络质量 QoS 分析、用户访问量分析统计等。例如,对云服务健康监护评估,对内监护服务器响应和网络状况;TCP、HTTP 应用层响应时间;TCP 的乱序问题、HTTP Status 情况等。对热点服务和网络进行扩容或者流量负载均衡,如图 6.4 所示。

图 6.5 给示出了一次 DDoS 攻击的情况,发生在某日凌晨 3 点。通过对归档化的流量进行分析,分析总结特征。

特征检测系统由高性能多核平台实现,可以基于任务的负载均衡调度,对安全策略灵活配置,可任意组合策略。通过 DDoS 攻击模式的分析采样,可以对新发生的 DDoS 进行模式识别与分类。特征检测匹配成功后,即刻启动流量牵引及流量清洗,实时发现攻击

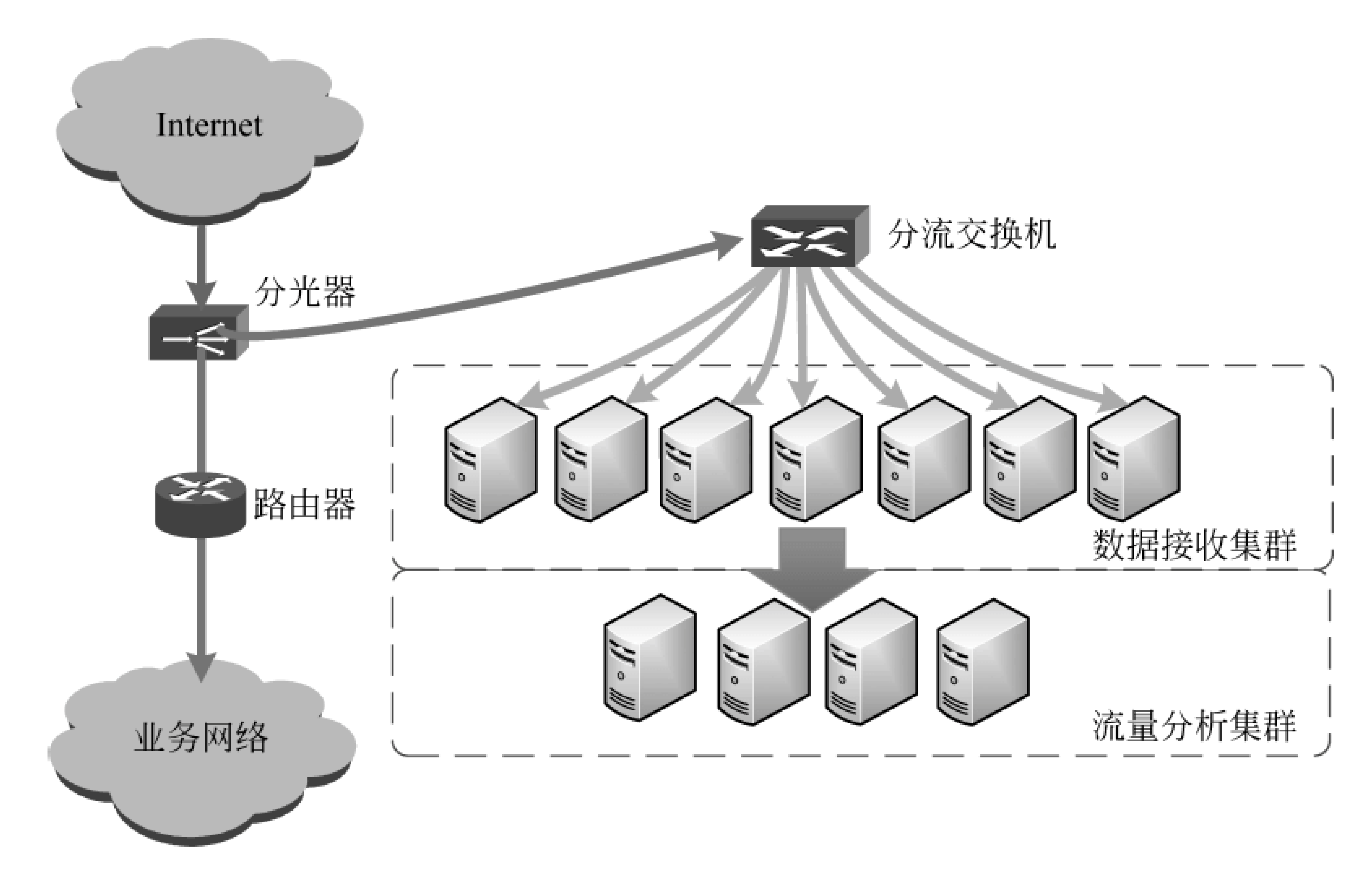


图 6.3　流量镜像系统

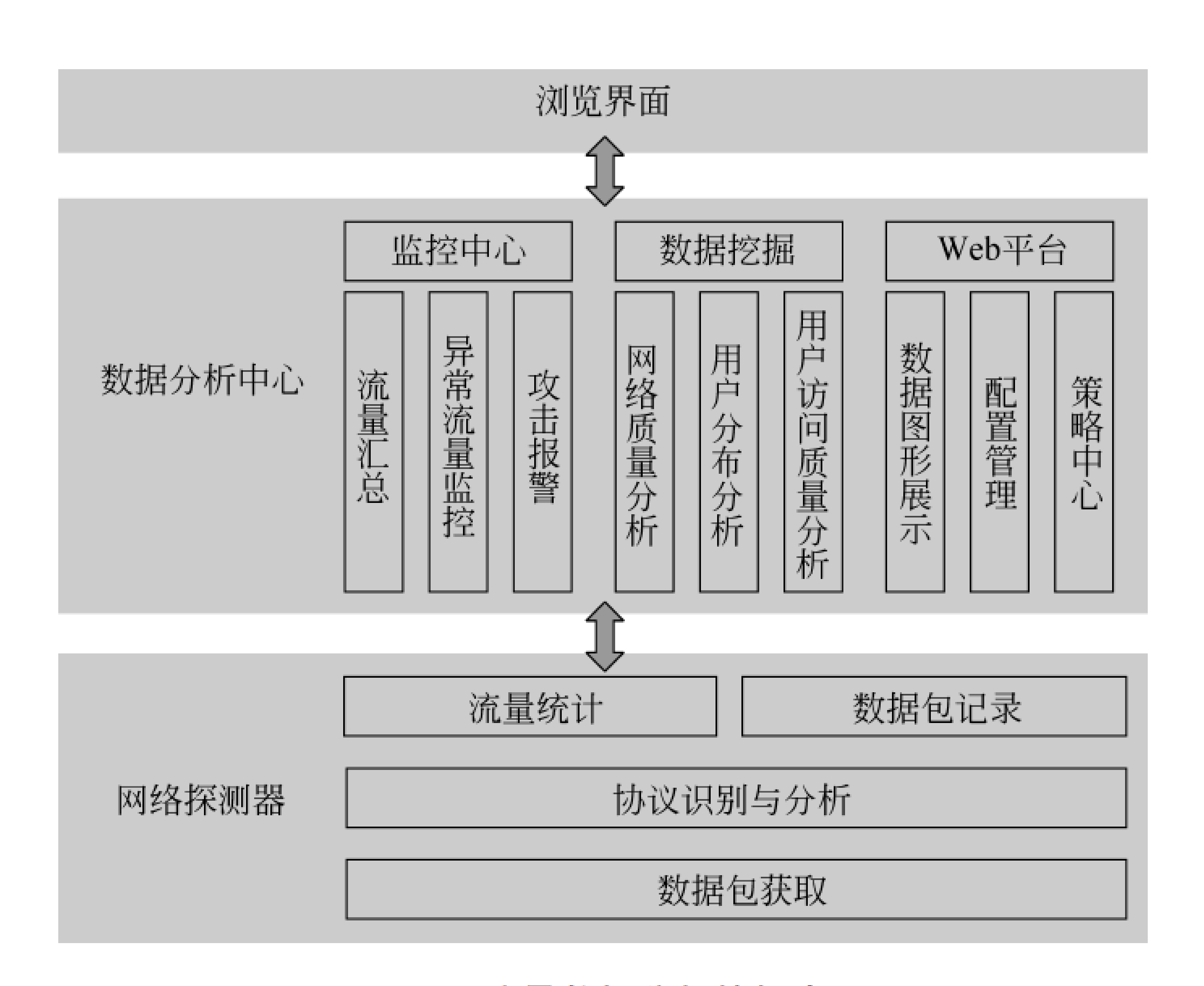


图 6.4　流量数据分析挖掘中心

特征，截住攻击流量。DDoS 攻击流量牵引与清洗如图 6.6 所示。

超大规模的数据处理能力问题，主要依赖于大数据存储与检索系统、精准可靠的异常流量识别算法以及高效特征匹配算法。

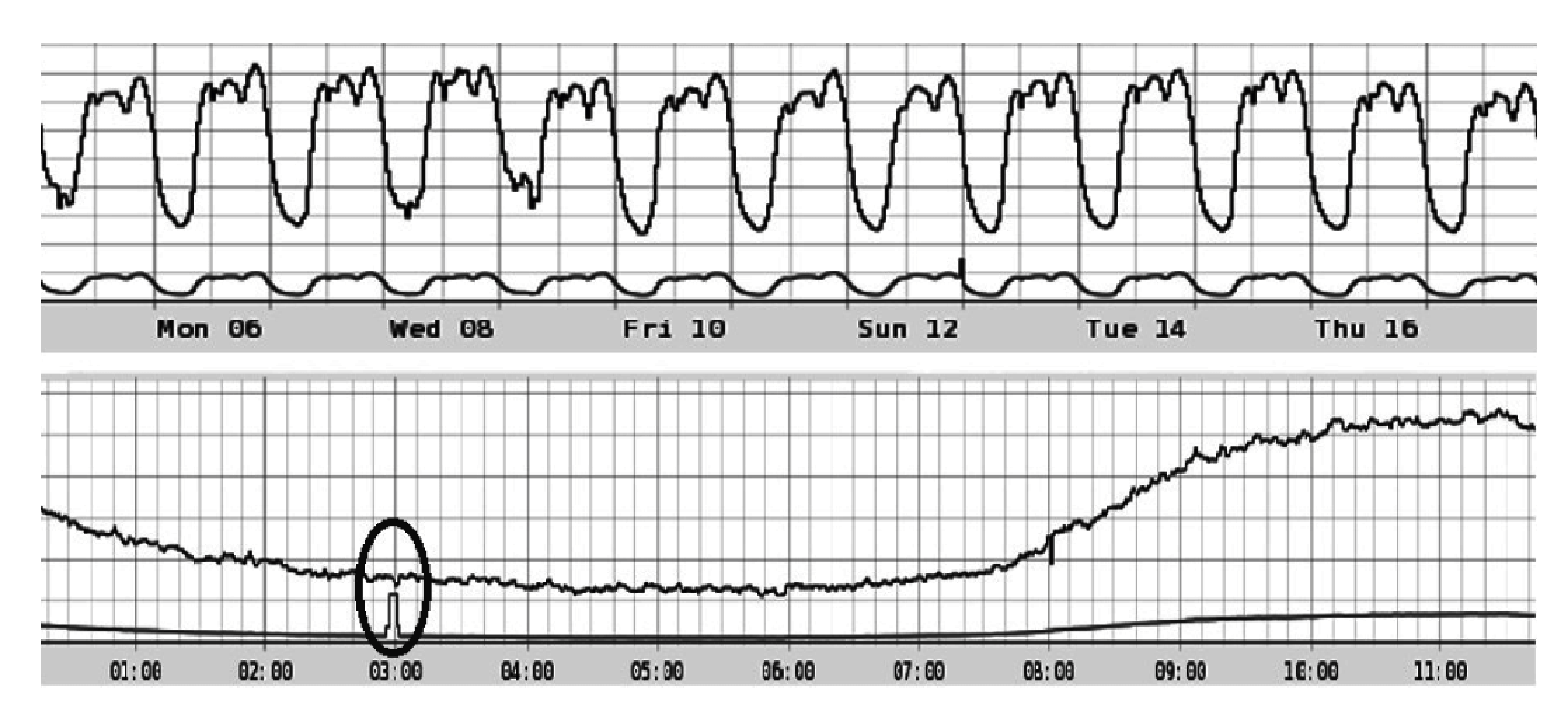


图 6.5　DDoS 攻击检测

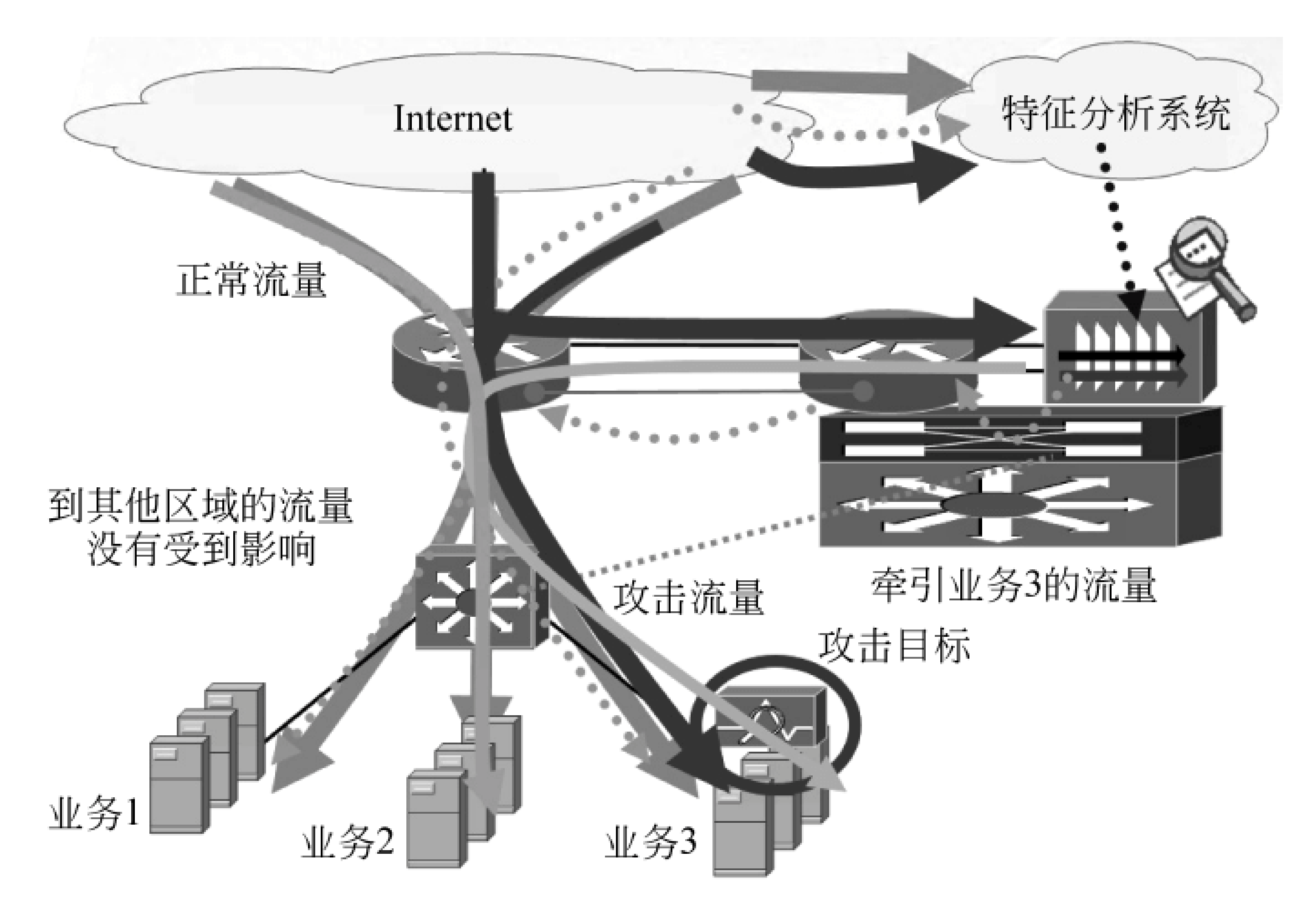


图 6.6　DDoS 攻击流量牵引与清洗

6.4　互联网安全实时对抗

在网络层面，数据中心一般都会部署 Snort 入侵检测系统，通过 Snort 监控出入流量，奇虎公司的万兆 Snort 入侵检测系统部署方案如图 6.7 所示。实现功能有万兆级网包获取，网流的大数据存储与计算，多数据中心汇聚，通过 TCP 协议 RST 信号实现旁路

阻断等。

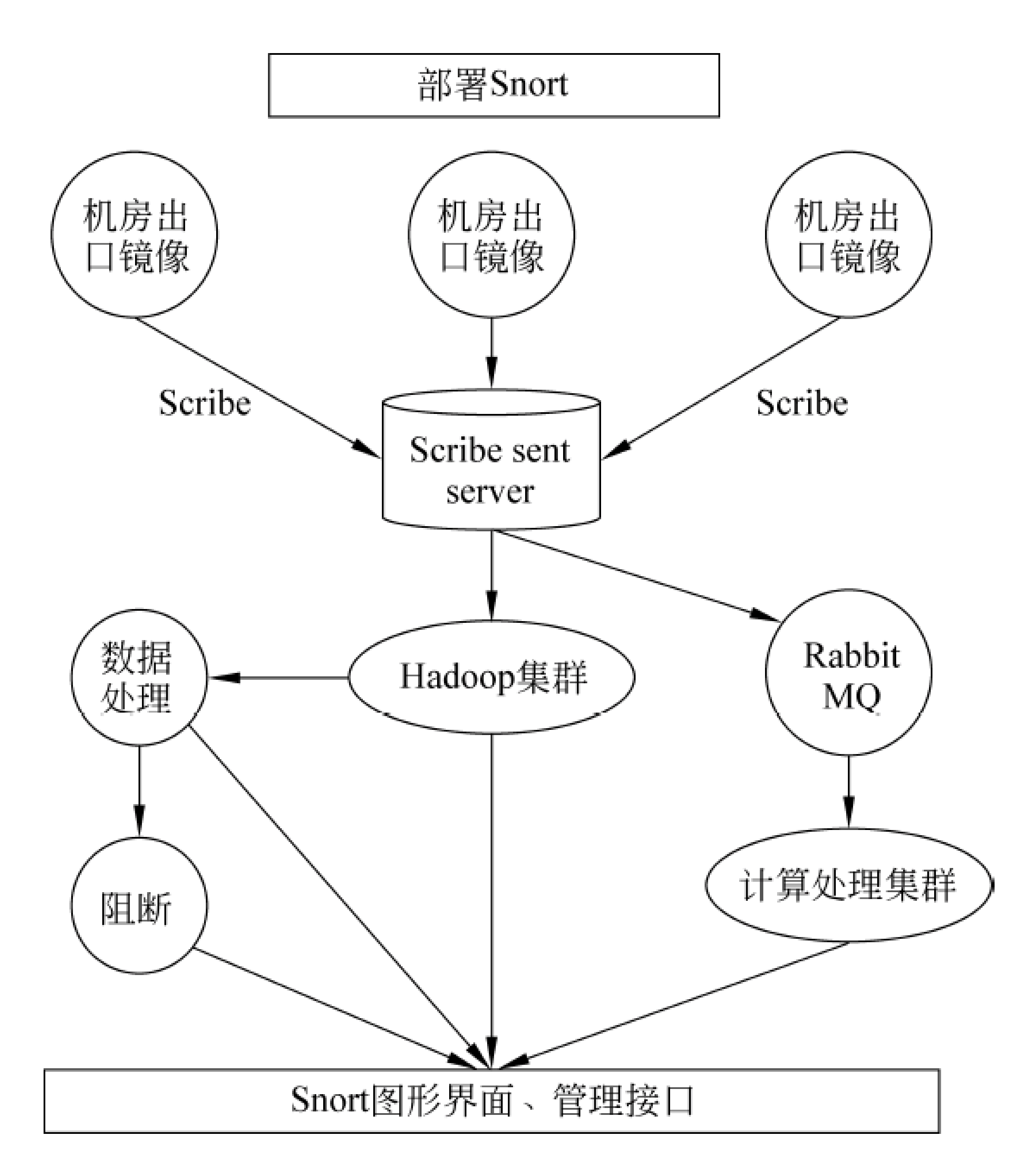


图 6.7　万兆 Snort 入侵检测

在数据中心 Web 服务前端，一般都会部署 Web 应用防火墙 WAF，通过 WAF 监控出入 Web 的流量，对 Web 攻击识别与旁路阻断。旁路 Web 实时监测系统部署方案如图 6.8 所示。

首先对 Web 访问数字化，同一 Web 日志格式如下：

```
IP -- [time] method uri protrol retcode len ref user-agent " domain x-forward-ip
```

其次，识别 Web 异常行为分类，常见 Web 攻击有 XSS 跨站攻击、SQL 注入攻击、文件包含攻击、Webshell 访问和敏感信息探测等。

这里，为了能够实现攻击的快速识别，使用了最新流式计算的 Twitter Storm 平台。实现万兆级网包获取与 HTTP 流重组，通过机器学习与规则提取，建立威胁模型，自动发现与提取攻击特征，识别一些 APT 行为(高级持续攻击行为)。

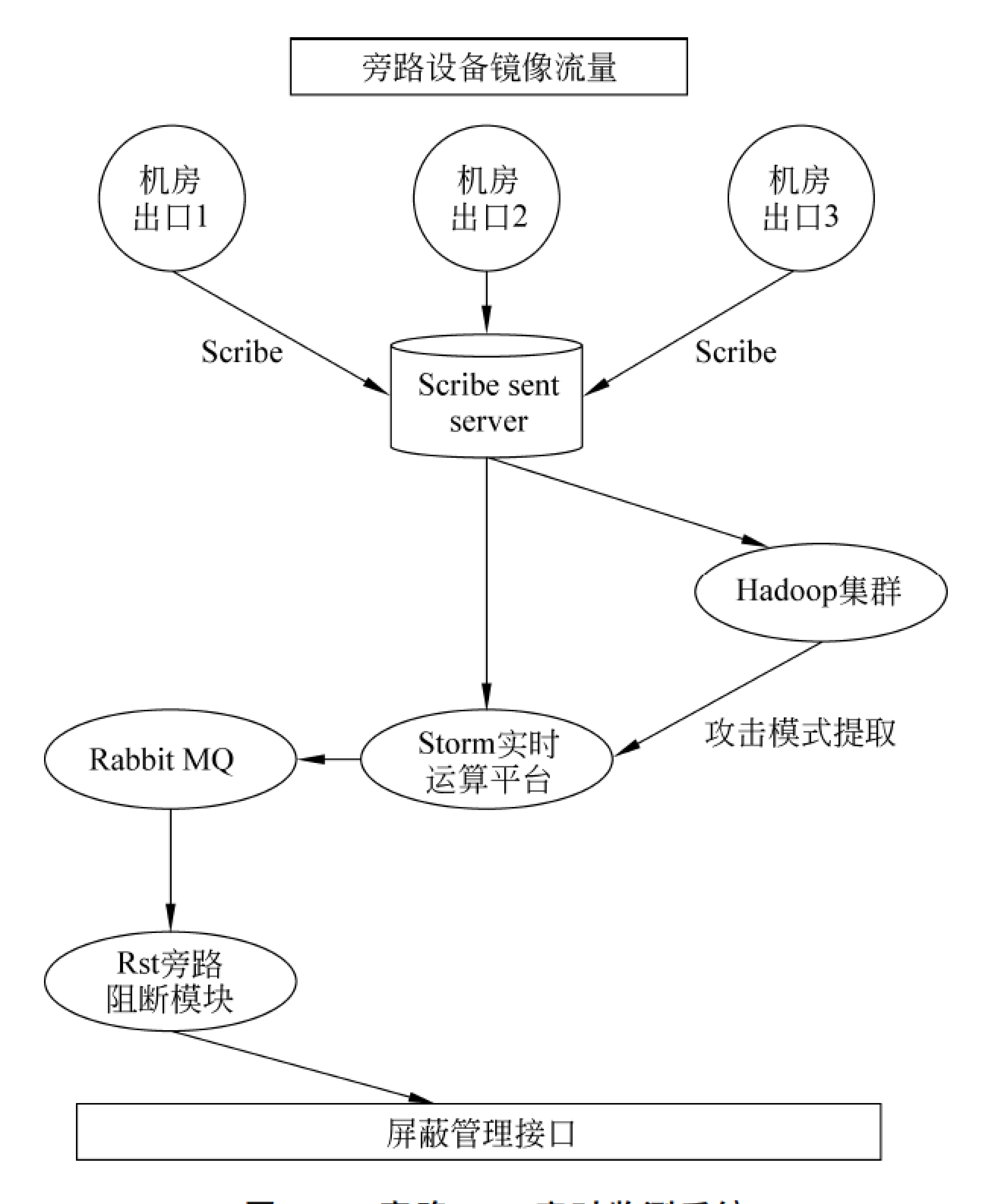


图 6.8　旁路 Web 实时监测系统

6.5　网络攻击检测与流式处理

在传统的数据处理流程中，总是先收集数据，然后将数据放到 Database 中。但是数据总是不断地产生，需要根据当前的数据实时地作出判断，因此流式计算（Stream Computing）解决大数据的时效处理。

在数据实时采集方面，完整地收集到所有安全设备日志数据，为实时应用提供实时数据。目前，日志数据采集工具有 Scribe、Kafka、Flume、TimeTunnel、Chukwa 等，均可以满足每秒数百 MB 的日志数据采集和传输需求。

对于流式处理，目前常见流式计算系统有 Twitter Storm 和 Yahoo！S4。其他高效处理平台还有 Facebook Puma3 和 Apache Spark 等。

6.5.1　Twitter Storm 流计算

Storm 是一个开源分布式的、容错的实时计算系统。Storm 建立在 Zookeeper 的基

础上，大量系统运行状态的元信息都序列化在 Zookeeper 中。Storm 用于处理消息和更新数据库（流处理），在数据流上进行持续查询，并以流的形式返回结果。

与 Hadoop 的批处理不同，Storm 采用可靠流处理，应用场景包括实时分析、在线机器学习和连续计算等。

6.5.2　Yahoo! S4 分布式流计算平台

S4（Simple Scalable Streaming System）最初是由 Yahoo!为提高搜索广告有效点击率的问题而开发的平台，是一个分布式流计算（Distributed Stream Computing）的模型，用于通过统计分析用户对广告的点击率，排除相关度低的广告，提升点击率。

S4 的设计目标如下。

（1）使用去中心的对等架构，所有节点提供相同的功能和职责，简化了部署和维护。

（2）通过在每个处理节点使用本地内存，避免磁盘 I/O 瓶颈达到最小化延迟。

（3）使用可插拔的架构，使设计尽可能地既通用又可定制化。

（4）提供简单的编程接口来处理数据流，友好的设计理念，易于编程，具有灵活的弹性。

（5）可以在普通硬件之上扩展高可用集群。

6.5.3　Facebook DataFreeway/Puma3

Data Freeway 是 Facebook 基于 Hadoop/Hive 的日志近实时分析系统，通过 Scribe-HDFS 接口，由 Continuous Copier/Loader 写入 HDFS 的 Datanode，对外统一用 Parallel Tailer，如图 6.9 所示。

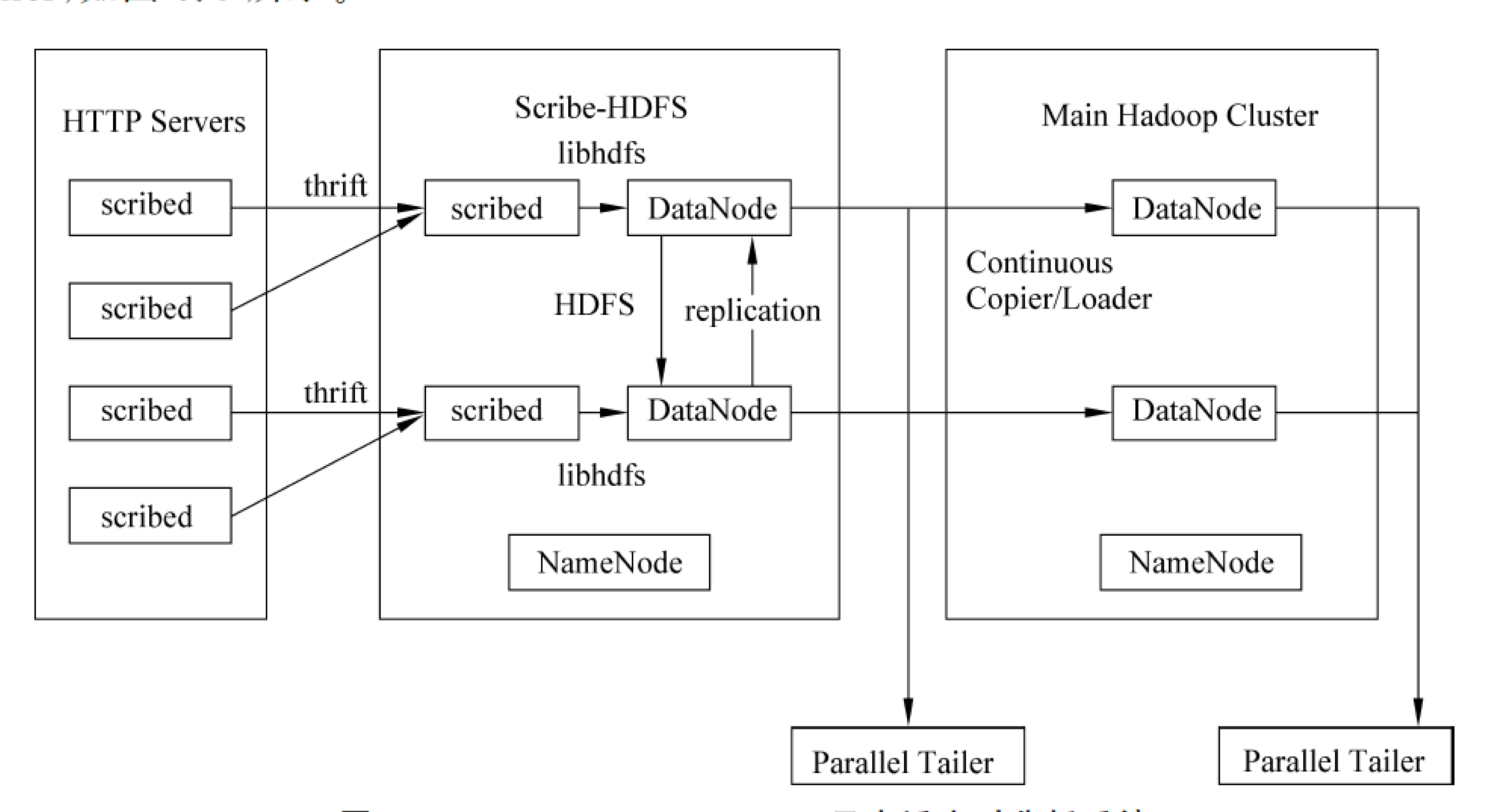


图 6.9　Facebook Data Freeway 日志近实时分析系统

Facebook Puma3 实时聚合和存储系统由 Puma2 发展而来，由 HBase 作为持久化 key-value storage，通过聚合 key Shard 作为 hashmap 存储在内存中，如图 6.10 所示。

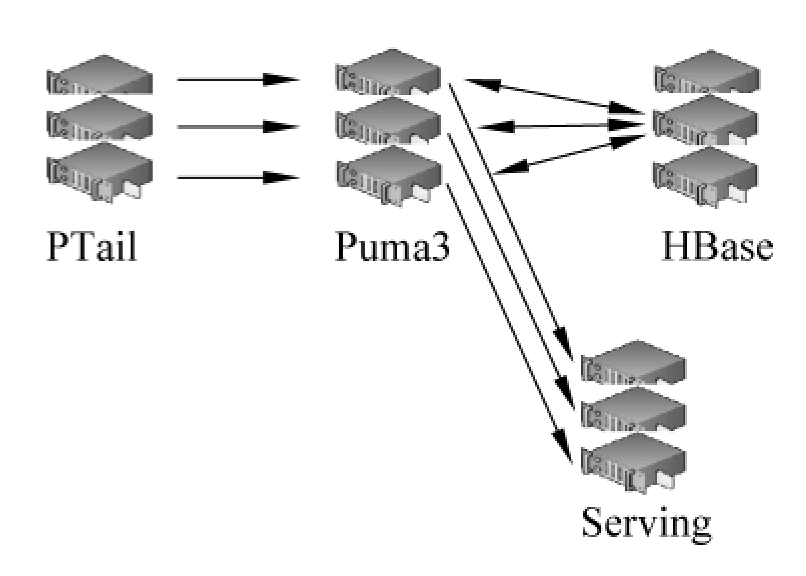


图 6.10 Facebook Puma3 实时聚合和存储系统

6.5.4 Apache Spark 平台

Apache Spark 由 Berkeley AMP 实验室 Ion Stoica 教授主导开发，是 BADS (Berkeley Data Analysis Stack)的工具之一。Spark 支持内存计算，加快速度，并提供了 Scala、Java 和 Python 编程接口。Spark 是支持 Shark 数据查询系统的引擎，Shark 是和 Hive 兼容的数据仓储系统，但是比 Hive 具有更快的查询速度。Spark 功能组件图如图 6.11 所示。

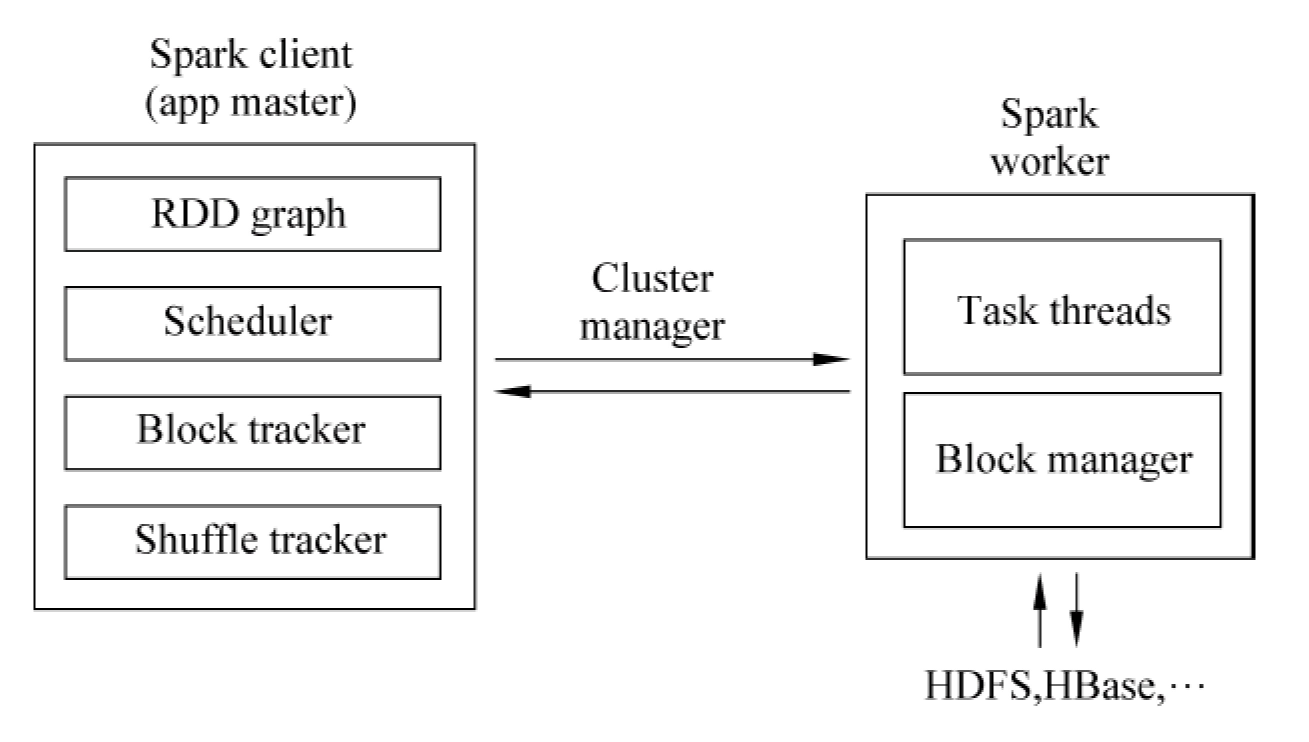


图 6.11 Spark 平台功能组件图

参 考 文 献

[1] 范承工，周宝曜，刘伟. 大数据. 战略，技术，实践[M]. 北京：电子工业出版社，2013.
[2] 刘鹏. 云计算(第 2 版)[M]. 北京：电子工业出版社，2011.
[3] 谭晓生. 基于大数据的网络安全攻击检测. 中国计算机大会，2013.

[4] 唐锡南. 个人超级计算时代的挑战. 清华大学信息技术研究院报告,2013.
[5] 陈震. 互联网安全原理与实践. 北京：清华大学出版社，2013.
[6] Apache S4 Distributed StreamComputing Platform. http://incubator.apache.org/s4.
[7] Twitter Storm. http://storm-project.net.
[8] Apache Spark. Lightning Fast Cluster Computing. http://spark.incubator.apache.org.

第7章

互联网化软件流量行为分析

互联网逐渐融入人类社会的方方面面，极大地促进了全球化的进程。同时随着互联网规模的扩大，越来越多的互联网公司开始组建自己的数据中心，将上万台计算机甚至几十万台计算机连接成一个云计算平台，通过云计算平台可以实现很多以前不敢想象的超大规模分布式计算。云计算平台与传统计算机相比较，为应用领域的难题解答实现了质和量的飞跃。为适应这种重大变革，软件系统进入一种新的形态：可演化、可连续反应以及可适应多目标等。软件实体之间可以进行协同工作，进行跨网的互联，从而发展为互联网环境下的新的软件形态——网构软件。

网构软件是互联网发展的产物，随着互联网的快速发展，用户隐私与用户数据安全也面临空前挑战，而软件的网络行为与用户安全息息相关：恶意程序多是通过联网行为，将窃取用户的隐私发送至互联网特定主机上。因此对网构软件的网络行为进行深度分析比较有价值。最近几年比较普及的安全卫士软件都属于网构软件。安全卫士软件都拥有联网云查杀模块，通过上传用户计算机上的可疑文件，在后端云服务器上进行查杀分析。安全卫士软件都有大量的联网行为，因此本章中作者将带领读者一步步深入分析国内最具代表性的4款安全卫士软件。

7.1 安全软件简介

我们选取4款最具影响力的安全卫士软件，并分别以中国的四神兽：青龙、白虎、朱雀、玄武为其命名。从4款安全卫士软件的官方主页功能介绍可以看出，都具有如表7.1所示功能模块。

表 7.1　网络安全软件的功能模块

模 块 名 称	模块功能描述
软件管理	软件升级、软件卸载、软件管理等功能模块
安全防护	系统修复、杀毒、修复漏洞、实时防护、下载保护、顽固木马克星、安全沙箱等
系统清理	清理垃圾、清除痕迹、清理插件
计算机加速	一键优化、开机时间管理、启动项、服务项、计划任务等
网络管理	管理网络流量、测试网速、网速保护、ARP 防火墙等

7.2　测 试 方 案

针对以上模块可能产生的联网行为，我们使用 4 台 Windows 7 虚拟机分别安装青龙、白虎、朱雀、玄武 4 款卫士软件进行测试。在测试过程中保证测试机器硬件、软件环境完全相同，并且 4 款安全卫士软件安装以后均未进行任何设置。为避免系统刚启动，安全卫士软件因进行扫描等活动可能产生的网络行为差异，我们在虚拟机开机 11h 以后(系统运行稳定，读者也可以选择在开机 2h 以后进行测试)，在相同的时间点开始抓取网络数据，并且在相同的时间点结束测试。

本次测试主要针对无用户干预的情况下安全软件自发产生的网络行为进行分析。通过安全网关数据采集模块记录所有的 Windows 7 虚拟机产生的网络流量(读者也可以通过 Windows 7 虚拟机上安装的 Wireshark 软件，抓取所有通过虚拟机网卡的数据，抓取的内容与安全网关所采集的内容相同)。对获取的数据包进行对比分析，一步步深入分析 4 款卫士软件网络行为的深层内涵。

7.2.1　测试环境

硬件平台：4 台 HPZ220 工作站，分别安装 VMware vSphere 5.0 虚拟化平台。

软件平台：Windows 7 64 位操作系统以及 Untangle 9.41 64 位安全网关。Windows 7 虚拟机通过安全网关以 NAT 方式访问互联网。将 Windows 7 虚拟机和 Untangle 安全网关克隆 4 份，分别部署在 4 台工作站上，如表 7.2 所示。

表 7.2　测试虚拟机环境

服　务　器	IP 地址	存 储 盘 阵	安全网关 IP	安 装 软 件
Server1	192.168.2.166	Datastore	*.*.134.135	青龙
Server2	192.168.2.166	Datastore6	*.*.134.137	白虎
Server3	192.168.2.166	Datastore4	*.*.137.227	朱雀
Server4	192.168.2.166	Datastore	*.*.134.136	玄武

网络部署如图 7.1 所示。

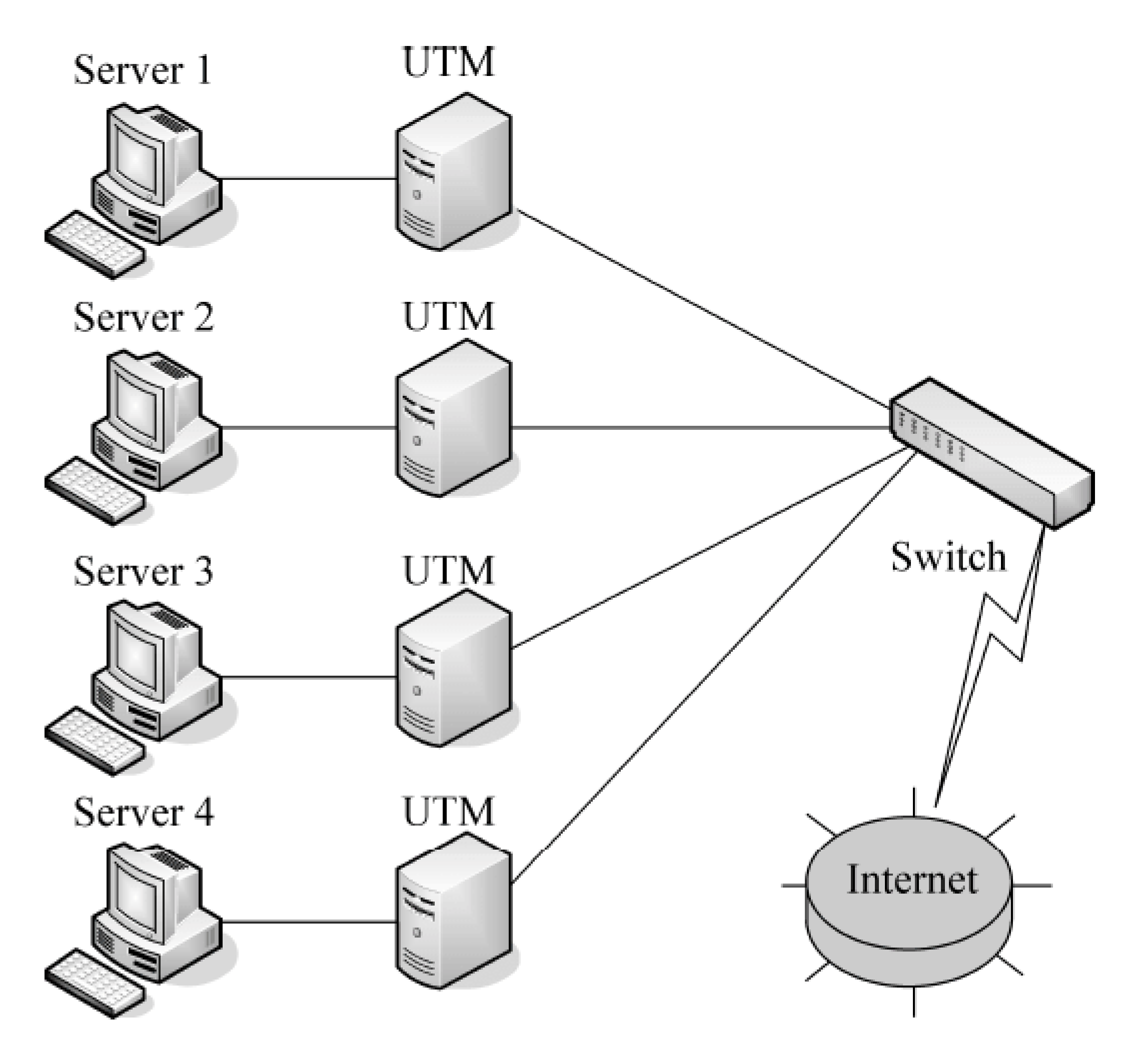


图 7.1 网络测试环境部署图

7.2.2 测试方法

主机测试之前同步系统时间,禁用自动更新服务,禁用自动休眠服务,取消屏幕保护。同时安装 4 款安全管理软件,安装完成后不运行安全体验,不运行漏洞修复,不采取任何操作。开机 11h 后开始抓取数据包(避免由于刚开机,安全软件进行模块升级、木马查杀等因素造成的测试误差),测试 24 小时。对抓取的 24 小时数据包进行分析。

7.3 网络流量分析

在我们的测试环境中,四台虚拟机通过 NAT 方式访问网络,NAT 对 UDP 不是很友好,所以我们以 TCP 数据为样本,分析 4 款卫士软件的网络行为。

使用 Wireshark 打开要分析的格式为 pcapng 的数据包文件,在 Filter 过滤窗口中输入 TCP,单击 Apply 按钮即可以过滤出所有的 TCP 数据包,如图 7.2 所示。

7.3.1 流量包进行统计

对之前抓取到的所有 TCP 数据包进行统计分析,使用 Statistics 下面的 Summary 一项,可以看到 Wireshark 软件对 TCP 数据包的统计信息,如图 7.3 所示。

对 4 款软件分别统计,将结果汇总成表 7.3。

pcapng [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp　Expression... Clear Apply Save

No.	Time	Protocol	Length	Info
3	0.00363300	TCP	66	49159 > http [SYN] Seq=0 Win=8192 Len
4	0.20898900	TCP	66	http > 49159 [SYN, ACK] Seq=0 Ack=1 W
5	0.20905600	TCP	54	49159 > http [ACK] Seq=1 Ack=1 Win=65
6	0.20925600	HTTP	271	GET /msdownload/update/v3/static/trus
7	0.20943200	TCP	60	http > 49159 [ACK] Seq=1 Ack=218 Win=
8	0.37930000	TCP	545	[TCP segment of a reassembled PDU]
9	0.59159500	TCP	54	49159 > http [ACK] Seq=218 Ack=492 Wi
10	0.96948600	TCP	1514	[TCP segment of a reassembled PDU]
11	0.96948700	TCP	1514	[TCP segment of a reassembled PDU]
12	0.96948800	TCP	1514	[TCP segment of a reassembled PDU]
13	0.96953800	TCP	54	49159 > http [ACK] Seq=218 Ack=4872 W
14	0.97042300	TCP	1514	[TCP segment of a reassembled PDU]
15	1.10945500	TCP	1514	[TCP segment of a reassembled PDU]
16	1.10945700	TCP	1514	[TCP segment of a reassembled PDU]
17	1.10946300	TCP	1514	[TCP segment of a reassembled PDU]
18	1.10950400	TCP	54	49159 > http [ACK] Seq=218 Ack=10712
19	1.11043500	TCP	1514	[TCP segment of a reassembled PDU]
20	1.11043600	TCP	1514	[TCP segment of a reassembled PDU]
21	1.11046000	TCP	54	49159 > http [ACK] Seq=218 Ack=13632
22	1.24945600	TCP	1514	[TCP segment of a reassembled PDU]
23	1.24946400	TCP	1514	[TCP segment of a reassembled PDU]
24	1.24946500	TCP	1514	[TCP segment of a reassembled PDU]
25	1.24951200	TCP	54	49159 > http [ACK] Seq=218 Ack=18012

Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Hughes_00:00:00 (00:00:10:00:00:00), Dst: Vmware_af:2f:43 (00:50:56:af:2f:43)
Internet Protocol Version 4, Src: 192.168.2.166 (192.168.2.166),
Transmission Control Protocol, Src Port: 49159 (49159), Dst Port: http (80), Seq: 0, Len: 0

图 7.2　pcapng 的数据包文件分析界面

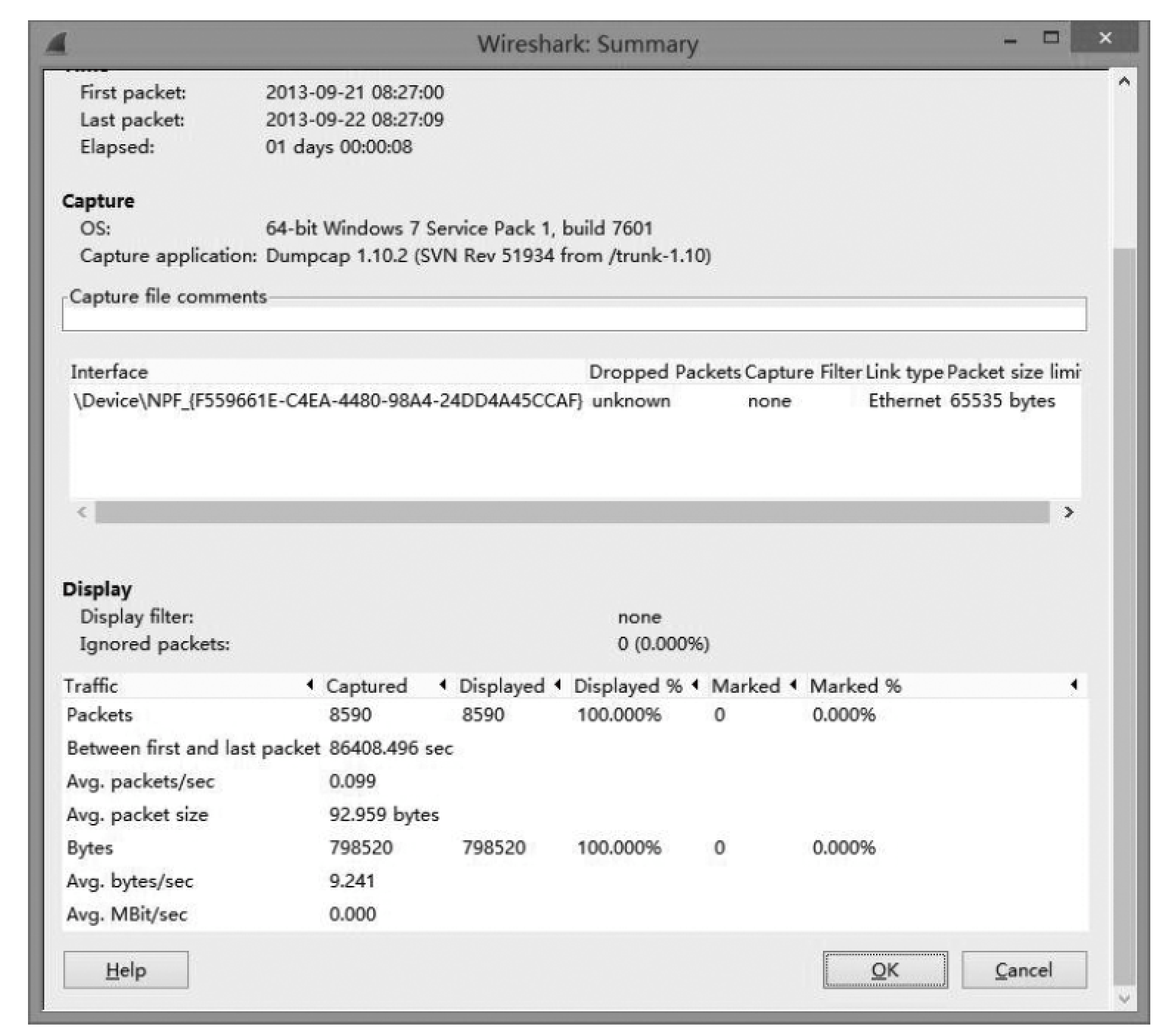


图 7.3　Wireshark 软件对 TCP 数据包的统计信息

表 7.3 4 款软件统计结果比较

	青 龙	白 虎	朱 雀	玄 武
Packets	8590	5743	38 388	12 680
Between first and last packet(sec)	86 408.496	86 401.308	86 414.360	86 475.348
Avg. packets/sec	0.099	0.066	0.444	0.147
Avg packet size	92.959	137.284	234.501	355.339
Bytes	798 520	788 423	9 002 020	4 505 697
Avg bytes/sec	9.241	9.125	104.173	52.104
Avg Mbit/sec	0.000	0.000	0.001	0.000

对其中 TCP 流量大小进行对比,可以得到图 7.4。

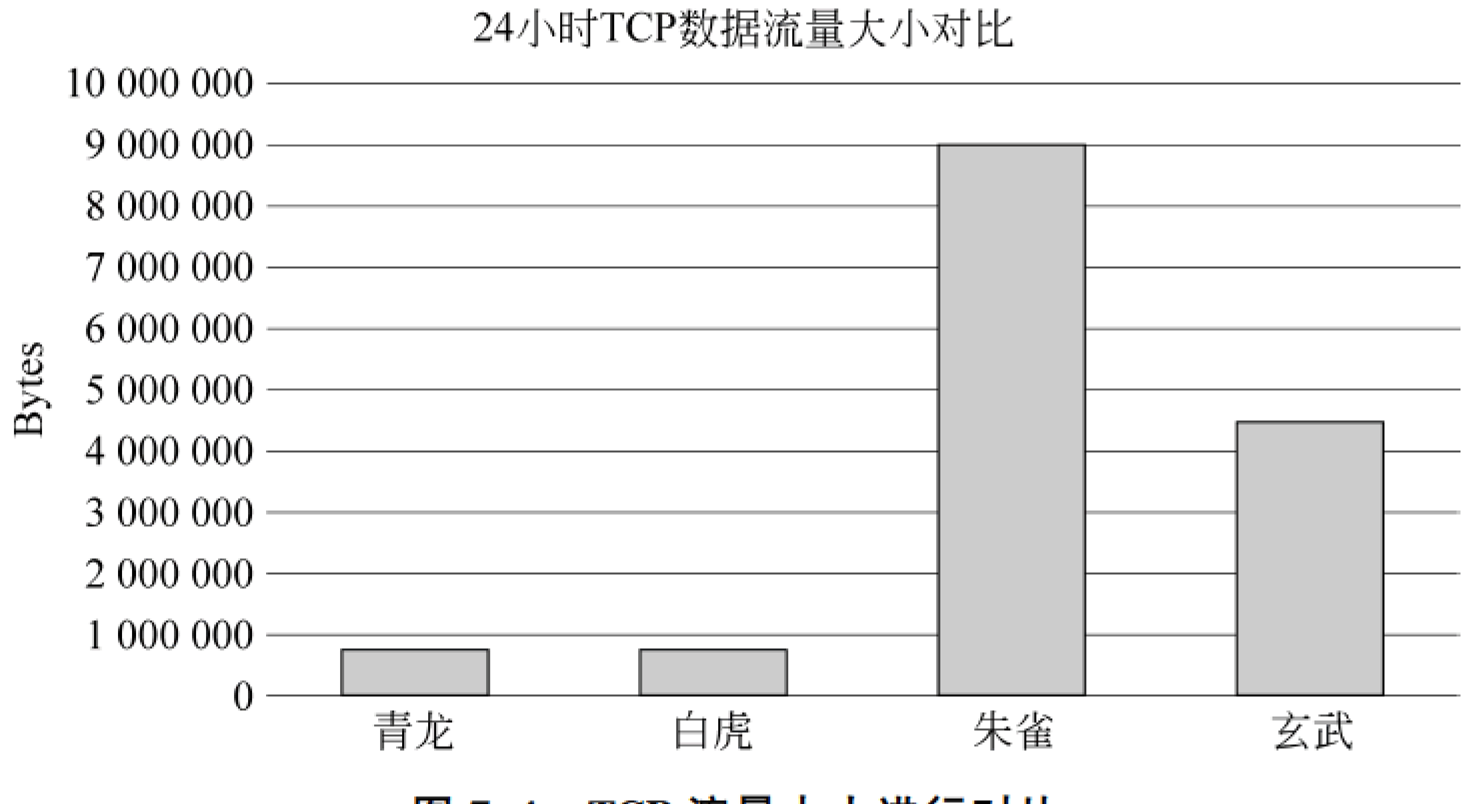


图 7.4 TCP 流量大小进行对比

可以看出,在 24h 测试期间青龙与白虎产生的 TCP 流量远少于另两款软件,分别是玄武的 1/5,朱雀的 1/10。

7.3.2 网络行为频率分析

充分利用 Wireshark 软件统计项中的 I/O Graph 功能,可以绘制出非常直观的流量分布图。下面我们先进行设置,Graph 1 是黑色标示,我们在后面对应的过滤框中输入 tcp,并在 Graph 2 红线的过滤框中输入 http,可以产生黑色和红色两条流量平滑曲线。再对横纵坐标范围进行设置:Tick interval 设置为 10min,Pixels per ticks 设置为 5。纵坐标使用 Bytes/Tick,最大范围为 20 000B。分别对 4 款软件的 TCP 和 HTTP 数据流量分布进行绘制,可以得到图 7.5~图 7.8。

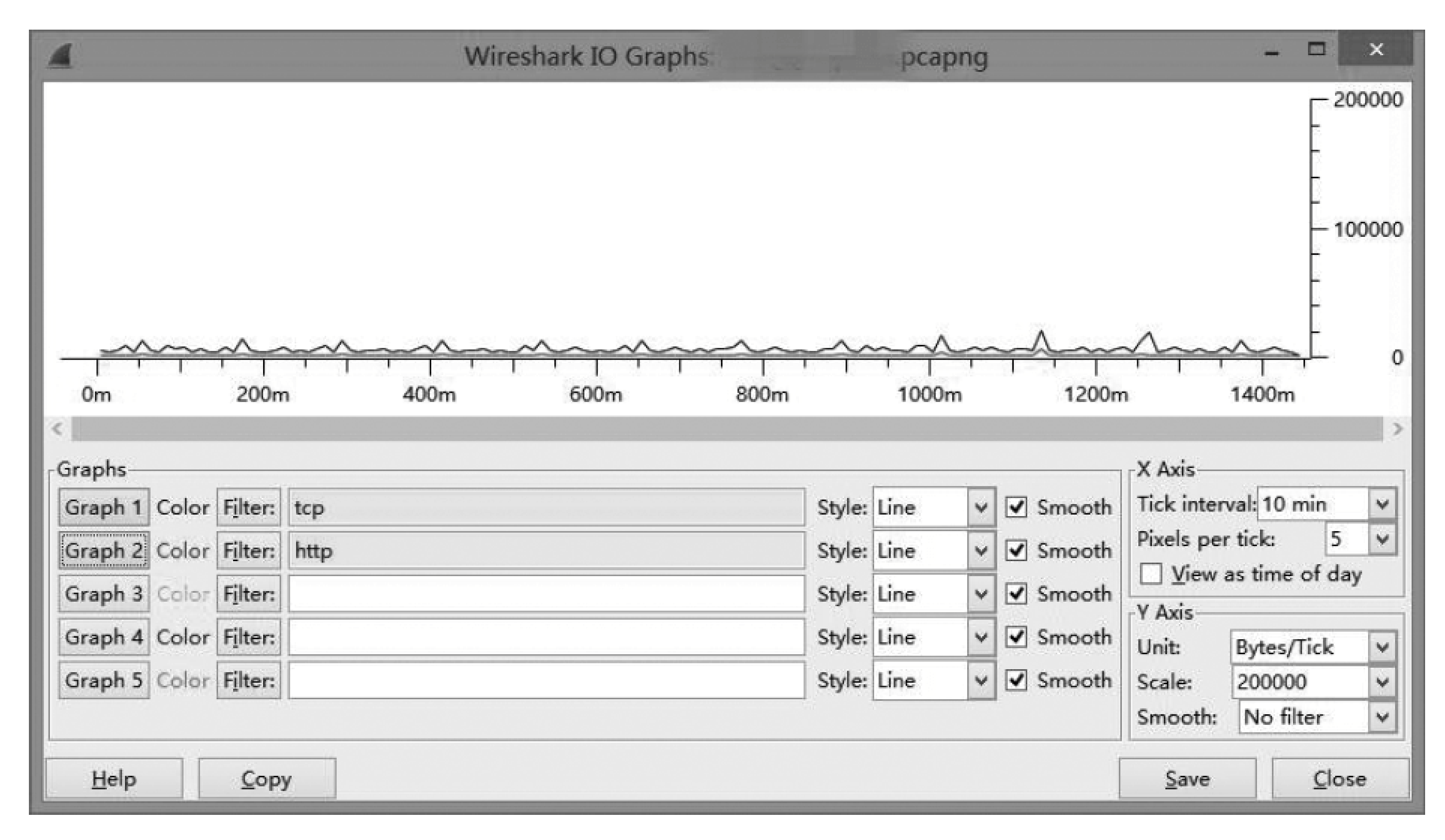


图 7.5　流量分布图(一)

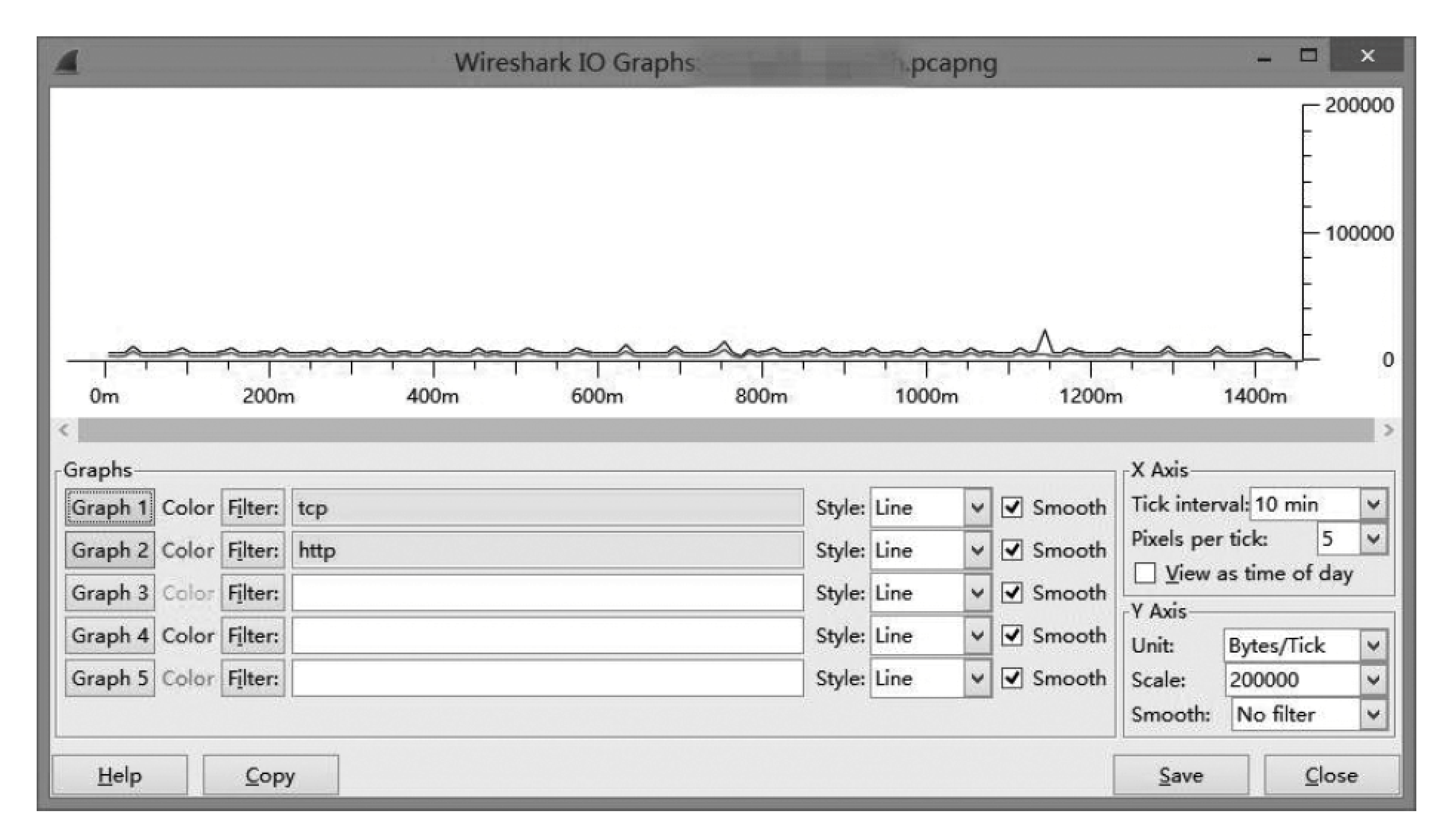


图 7.6　流量分布图(二)

曲线类型为 Line,即平滑曲线。所以上述四图可以给读者一个直观的印象,什么时间点会产生流量,以及产生多大的流量。24h 测试期间青龙和白虎所产生的流量非常小;360 安全管家产生了 4 次大峰值的 TCP 流量,约 6h 产生一次;玄武则产生了 24 次大峰值的流量,约 1h 1 次。下面我们再继续深入分析。

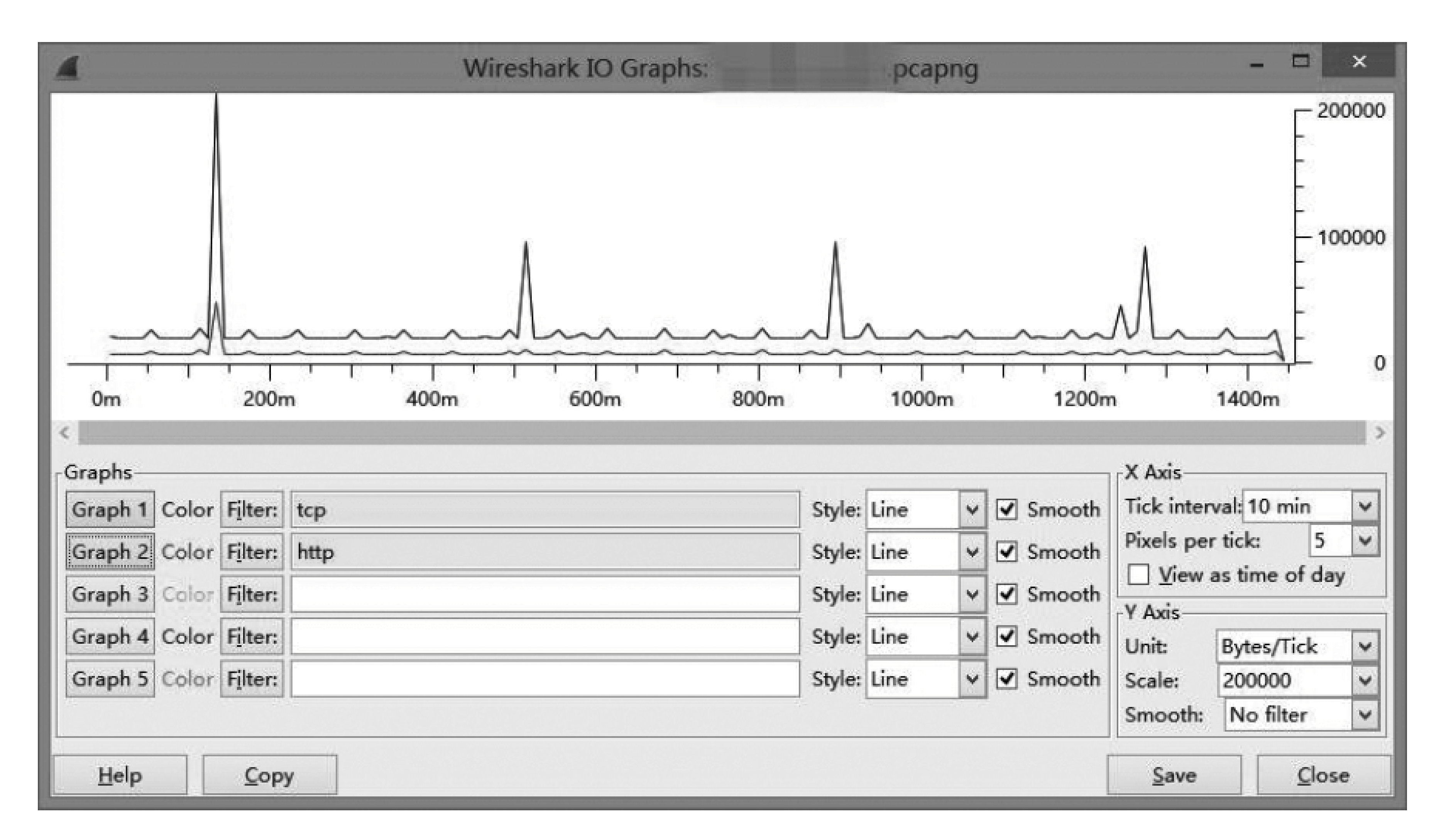


图 7.7 流量分布图(三)

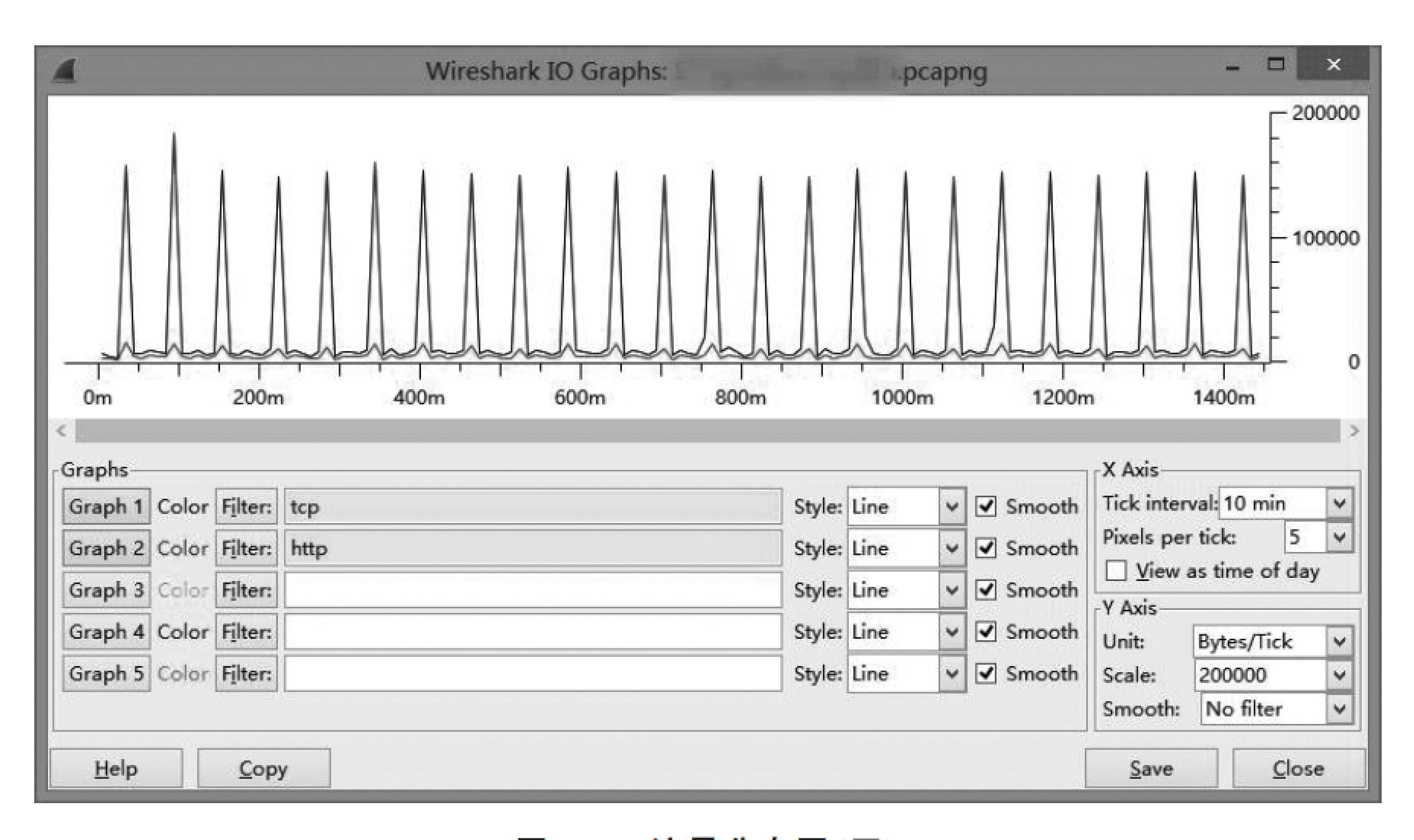


图 7.8 流量分布图(四)

7.3.3 网络行为数目分析

7.3.2 节全时域流量对比图是从全局角度给读者一个直观概念,什么时间会产生多大的流量,以及这些流量分布是否有规律可循。为了进一步方便用户了解这些安全软件的网络行为,根据 TCP 协议实现过程,如图 7.9 所示。

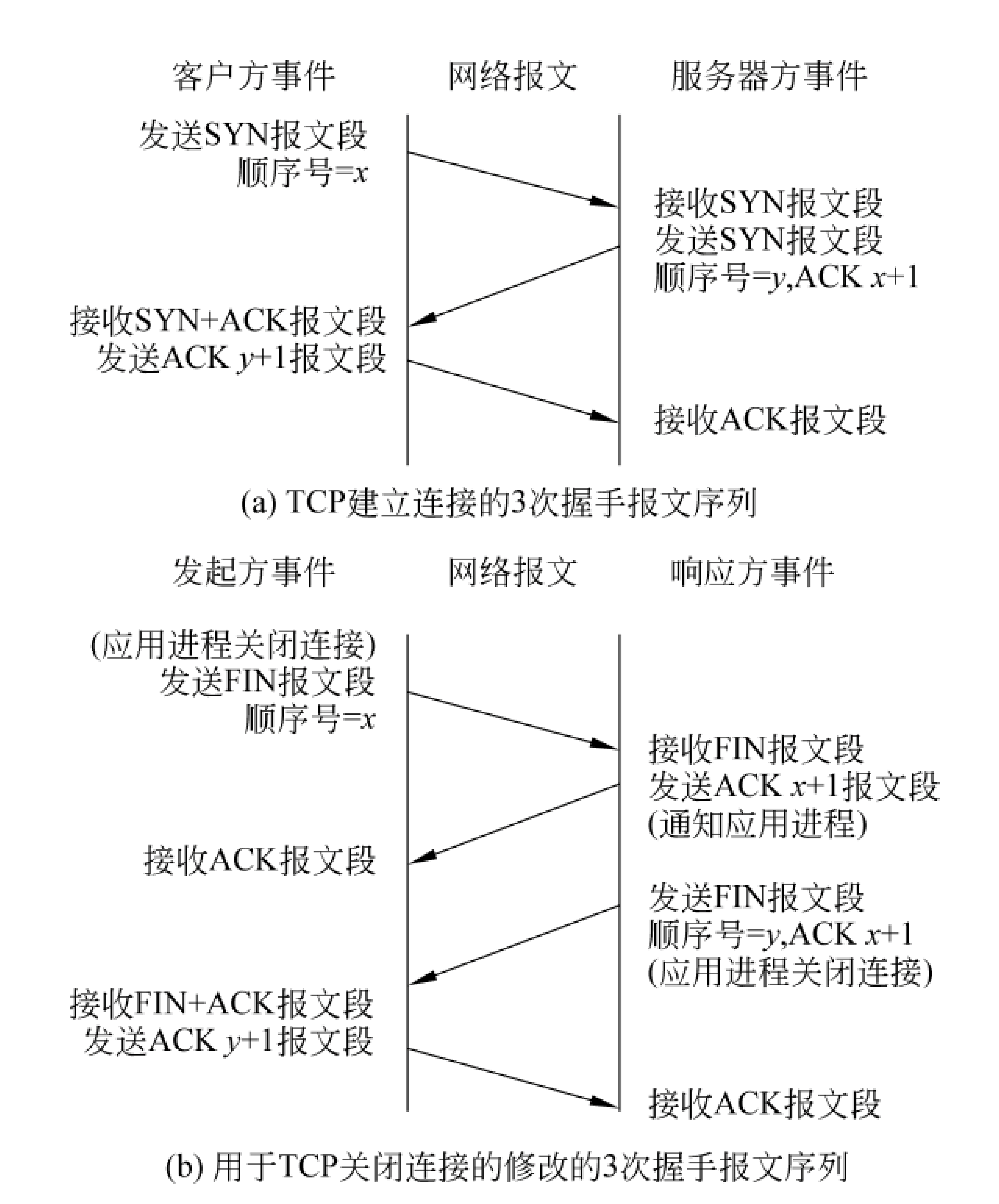


图 7.9　TCP 3 次握手报文序列

一次 TCP 活动需要 3 次握手以建立连接和 4 次挥手以结束连接。为了方便描述网络行为数目，可以将一次 TCP 连接从建立到结束的过程定义为一次网络行为。我们统计的安全卫士软件的网络行为如表 7.4 所示。

表 7.4　安全卫士软件的网络行为

卫士软件	网络行为数	卫士软件	网络行为数
青龙	738	朱雀	2951
白虎	348	玄武	963

从表 7.4 可以看出测试期间，4 款软件发起的网络行为次数差距比较大，青龙、白虎和玄武的网络行为数都少于 1000，而朱雀的网络行为接近 3000，意味着不到 30s 就会发起一次网络请求。而网络活动数最少的白虎平均 4min 才会发起一次网络请求。在这里读者会不会产生一个疑问：卫士软件是否需要如此频繁地访问网络呢？

由于青龙数据包通过 HTTPS 进行加密传输，白虎数据包仅仅为简单的状态确认并且每次数据包内容相同，因此未对这两款卫士软件进行详细分析。下面以朱雀和玄武为

例详细分析网络行为。

7.3.4 网络行为间隔累积分布对比

从 pcapng 数据包中读取两次相邻的网络行为时间间隔，使用累积分布函数可以画出 4 款软件的网络行为间隔累积分布图。如果想粗糙地获取一个模型进行分析，不需要编程，从图 7.5～图 7.8 入手，将纵坐标范围设小一些，横坐标单位值也设小一些，这样 Wireshark 可以帮人们统计更多的网络行为，以及发生时间，复制这些数据在 MATLAB 中进行分析，也可以得到一个直观的 CDF 分布图。

从图 7.10 可以看出，朱雀所有 TCP 流之间的时间间隔都集中在 30～40s，而其他 3 款软件 70%的 TCP 流都是集中出现的，即功能模块升级、特征库升级等网络活动大都是同时出现。而其他 TCP 活动之间间隔分布比较均衡，白虎的最大网络行为间隔达到了 10min。可见，安全卫士软件的联网频率不需要特别高，像朱雀这样的网络行为不是必需的。

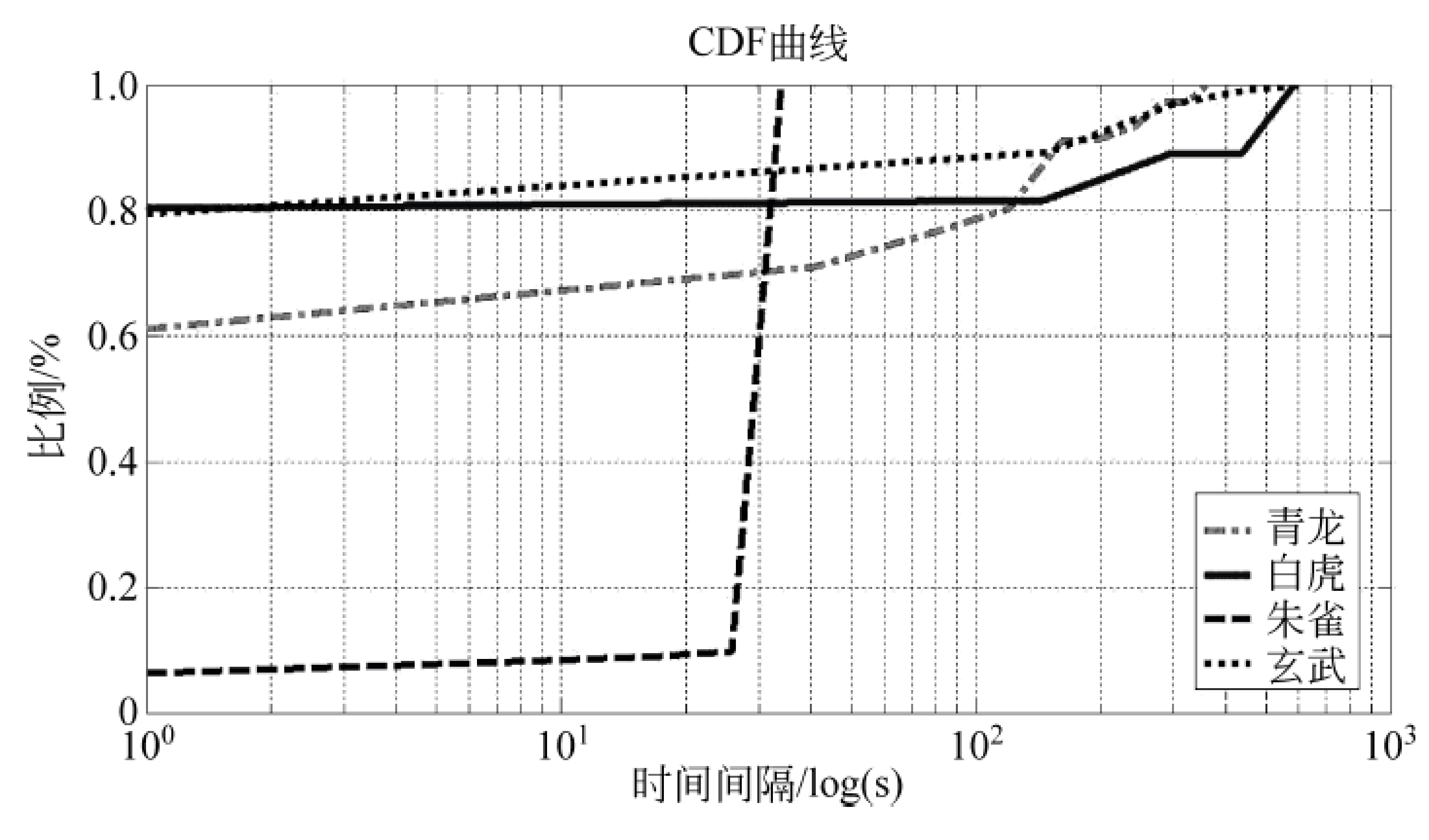


图 7.10 CDF 分布图

7.3.5 朱雀网络行为分析

因为朱雀的网络行为数据是明文传输的，所以朱雀分析比较清楚。我们可以从每次网络行为中的有效载荷 HTTP 中分析具体传输的数据。同样在 Filter(过滤)窗口中输入过滤参数 http 后点击应用，就可以得到实验期间朱雀所有的 HTTP 数据包。以其中一个数据包为例进行分析，如图 7.11 所示。

从 URL 字段可以获取我们想要的内容。直接打开这个链接可以发现 URL 页面为 OK 字符。同样，对其他 URL 进行下载分析，CAB 压缩包文件应该先解压再分析。将 CAB 解压以后的文件，对应朱雀的安装目录，在根目录下的文件为主功能模块文件及子

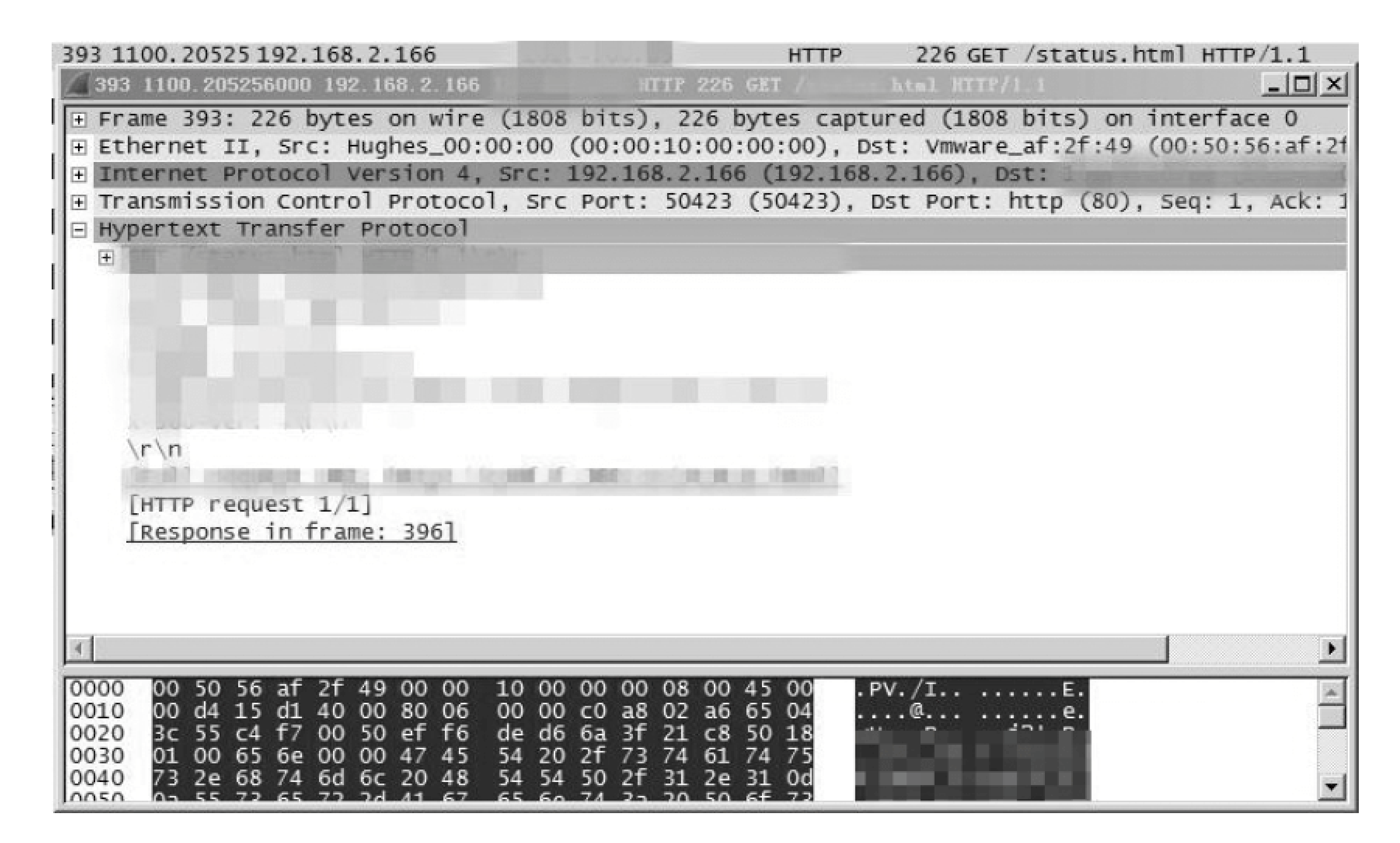


图 7.11　软件朱雀的流量数据包分析

文件，软件朱雀流量行为如表 7.5 所示。

表 7.5　软件朱雀流量行为

朱雀模块网络行为		
联网频率	功能描述	下载文件
6h 会产生一次	主模块升级	*.cab
	主模块升级	*.cab
63min 一次	主模块升级	*.cab
测试期间只发生一次性的网络活动	朱雀功能模块文件请求	*.dll
	朱雀功能模块文件升级	*.dll
	软件管家模块升级	*.dll
	主模块升级	*.dll
	主模块升级	*.exe
	主模块升级	*.exe
	主模块升级	*.dll
	垃圾插件特征	*.dat
	软件管家模块升级	*.ini

续表

朱雀模块网络行为		
联网频率	功能描述	下载文件
测试期间只发生一次性的网络活动	朱雀功能模块升级	*.dll
	朱雀功能模块升级	*.dat
	朱雀功能模块升级	*.dat
	朱雀功能模块升级	*.cab
	系统漏洞信息更新	*.dat
	未知	*.htm
	主模块升级	*.tpi
	朱雀功能模块	*.dat
	朱雀主模块升级	*.dll
	朱雀功能模块升级	*.dll
	未知	返回{"result":"-1","data":""}
	主题壁纸升级	*.png
	朱雀深度扫描模块	*.dll
	朱雀软件管家模块升级	*.dll
	朱雀软件管家模块升级	*.dat
	朱雀软件管家模块升级	*.ddll
上传文件分析：63min 一次	朱雀用户安全配置文件	* * *
	朱雀安全升级	*.php
云查杀—可疑文件扫描请求：30s 一次	定期向服务器发送状态请求	*.html
无法确定用途的 TCP 流		

测试期间根据网络访问频率可以看出，很多功能模块都是集中进行一次模块更新，部分模块更新更为频繁。其中有 3 种网络行为需要注意。

（1）上传文件行为。朱雀每隔 63min 会将用户安全配置文件和安全升级文件上传至服务器，虽然内容加密不能分析，但是从每次数据包内容都不同可以推断出每次上传的内容也都不同。这两种信息是否需要如此频繁地进行上传呢？还是应该在用户配置改变时进行上传更合理？

(2) 频繁向服务器发起朱雀云安全——扫描可疑文件请求,从服务器获取 status 状态字,status 内容为 OK,但具体用途也不得而知。

(3) 存在大量无法确定用途的 TCP 流,以编号为 282 的 TCP 流为例,其网络行为如图 7.12 所示。

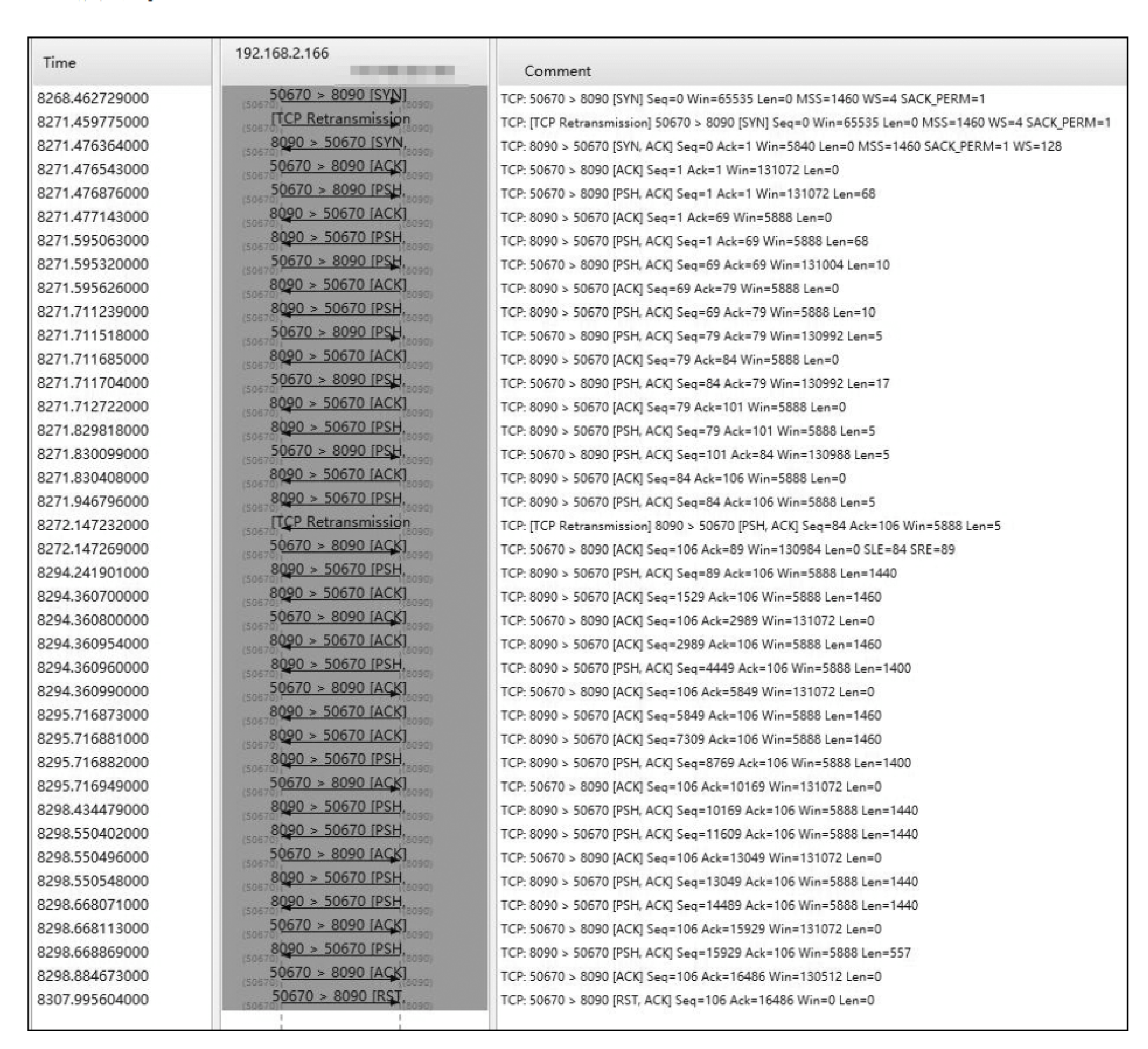


图 7.12　软件朱雀网络行为

特定通信端口无法确定具体用途。此类 TCP 流占总 TCP 流量的 7%。通信服务器分布在多个地区。

7.3.6　玄武网络行为分析

玄武流量行为如表 7.6 所示,其网络行为中内部数据更新节点也覆盖很多地区,具体如何实现无法确定。

表 7.6 软件玄武流量行为

玄武模块网络活动		
联网频率	功能描述	下载文件
10min 一次	玄武消息更新节点	*.pack
	玄武消息更新节点	new/*.pack
12min 一次	获取与软件版本相关的 CV 信息	cv.*
30min 一次	配置信息更新模块	*.dat
1h 一次	玄武公司内部数据更新节点	*.dat
	玄武公司内部数据更新节点	* * *
	玄武公司内部数据更新节点	*.dat
	玄武公司内部数据更新节点	*.dat
	消息推送节点	messenger *
	玄武消息快速更新节点	quick *
3h 一次	更新配置文件	* * l
	非安全模块的信息交换节点	* * *
15h 一次	玄武公司网盾更新	* * net
16h 一次	会员信息维护节点	*s.dat

7.3.7 远程通信地址分析

青龙在测试期间与 4 台腾讯服务器建立连接，如图 7.13 所示，其中流量最多的连接是与两台服务器通过 HTTPS 协议传输数据，由于内容加密而无法深入分析。其余网络行为是从另外两台服务器上获取 *.cgi 文件。

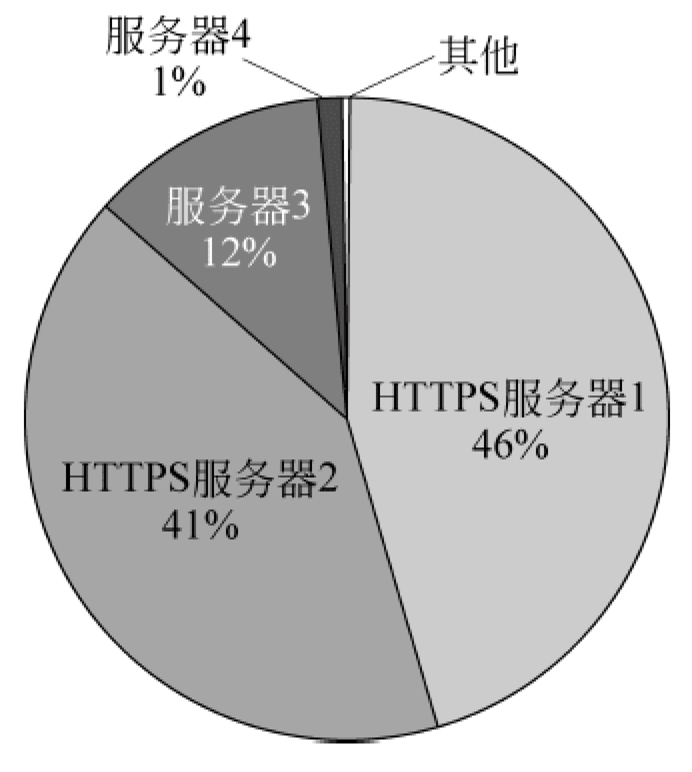


图 7.13 青龙远程地址聚类分析

白虎在测试期间访问了 3 台在联通网络上部署的 CDN 服务器，每一次网络行为都含有 3 次与服务器的交互活动，交互的数据包经比对发现内容相同，另外有少量访问微软服务器和检测系统升级的行为，如图 7.14 所示。

对朱雀通过远程地址按功能聚类分析发现，其在测试期间 62％的 TCP 流量都是由教育网内部服务器升级模块产生，30％的流量为朱雀的可疑文件扫描请求，7％

的 TCP 活动功能不能判定，其他网络行为在 TCP 流量中不到 1%，如图 7.15 所示。功能不能判定的远程主机如表 7.7 所示。

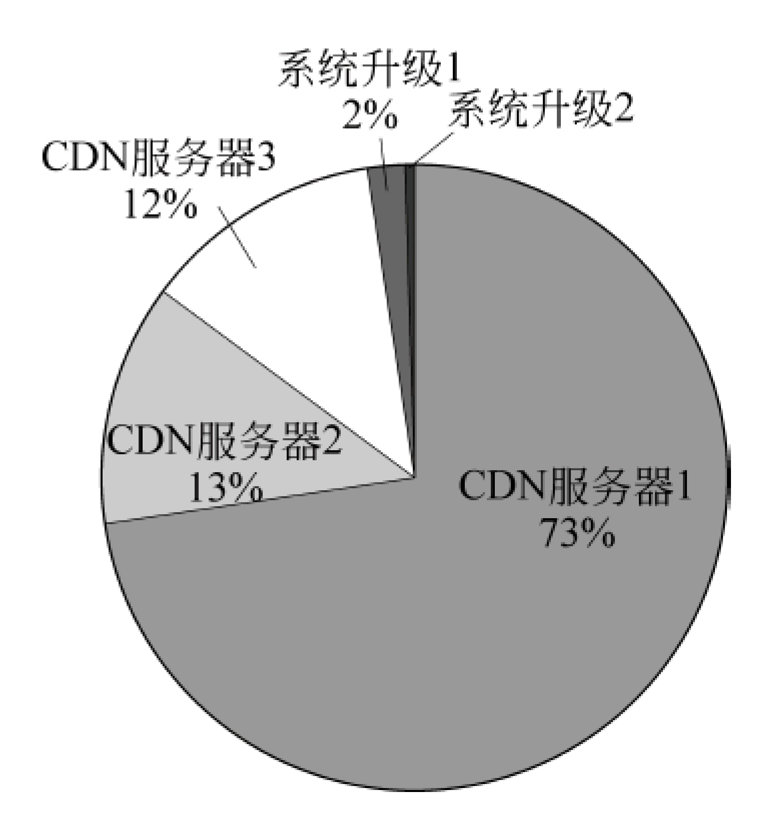


图 7.14　白虎远程地址聚类分析

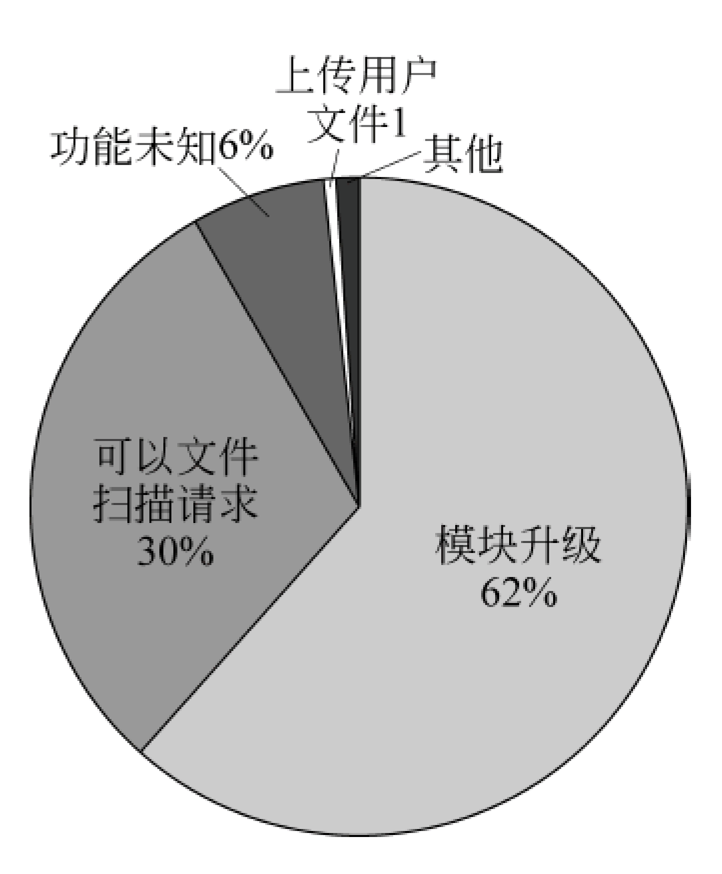


图 7.15　朱雀远程地址聚类分析

表 7.7　软件朱雀外联 IP 地址列表

IP	位　　置	IP	位　　置
1.30.*.*	内蒙古联通	119.166.*.*	山东省青岛市联通
101.18.*.*	河北省保定市联通	119.178.*.*	山东省菏泽市联通
101.4.*.*	朱雀在教育网内节点	119.55.*.*	吉林省长春市联通
101.4.*. *	朱雀在教育网内节点	123.234.*.*	山东省青岛市联通
110.228.*.*	河北省石家庄市联通	175.171.*.*	辽宁省大连市联通
110.242.*.*	河北省石家庄市联通	220.181.*.*	朱雀在教育网内节点
112.225.*.*	山东省青岛市联通	221.207.*.*	黑龙江省哈尔滨市联通
112.254.*.*	山东省青岛市联通	60.208.*.*	山东省济南市联通
114.250.*.*	北京市联通	60.21.*.*	辽宁省锦州市联通

这 18 个 IP 地址主要分布在华北和东北地区，用途目前无法通过流量分析获知。

玄武远程地址聚类分析如图 7.16 所示，玄武远程通信主机中 TCP 流量最多的功能模块也是同多台主机同时进行通信，如表 7.8 所示。

表 7.8 软件玄武外联 IP 地址列表

玄武内部数据更新节点	
113.5.*.*	黑龙江省哈尔滨市联通
123.130.*.*	山东省烟台市北京蓝汛通信技术有限责任公司联通 CDN 节点
218.29.*.*	河南省郑州市巩义市联通 ADSL
218.61.*.*	辽宁省大连市北京蓝汛通信技术有限责任公司联通 CDN 节点
221.192.*.*	河北省廊坊市联通
221.8.*.*	吉林省通化市 /梅河口市联通

玄武的通信特点是，同一功能模块会向多个地区发起请求，但是不能确定其升级采用的是 P2P 模式。玄武也有一些网络行为需要注意。

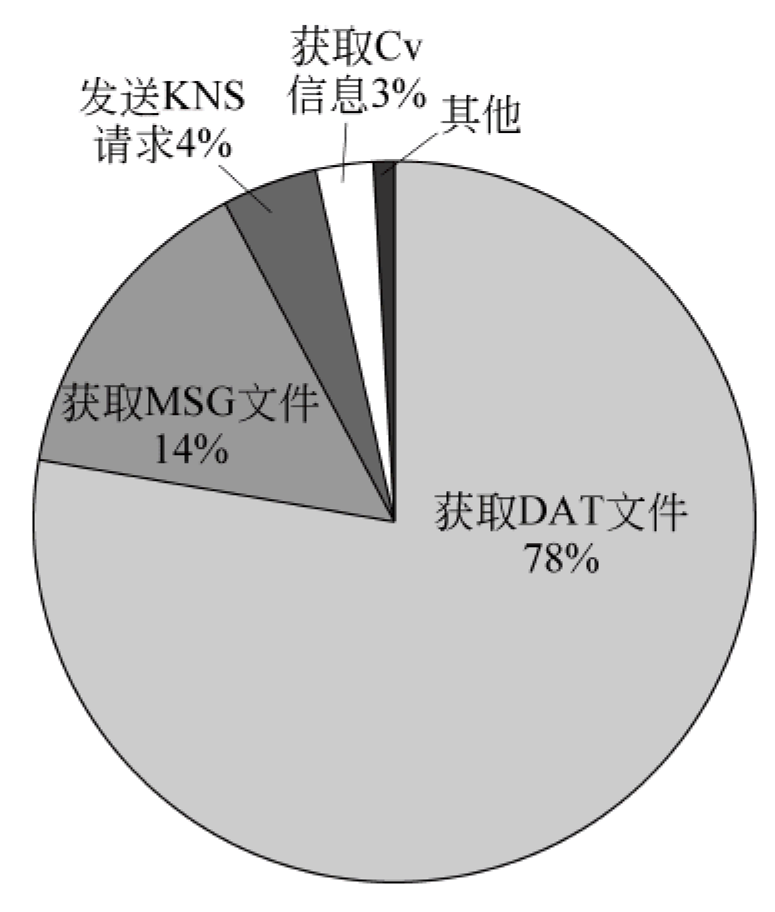


图 7.16 玄武远程地址聚类分析

(1) 非安全模块信息交互，玄武会推送给用户一些广告信息，如 wan.**.com 域名下的一些信息。

(2) VIP 信息推送，此网络行为出现在本身就含有广告过滤功能的安全卫士软件中是否合理？

通过对 4 款软件对比分析发现，网络行为各有特点：青龙的服务器都位于国内某省，联网升级需要连接至青龙公司部署在该省的服务器。白虎升级通过两个域名 d.**.com、s.**.com，这两个域名通过 CDN 技术，在国内部署了很多服务器，在各地访问时请求的地址因所在区域不同而不同。朱雀虽然在教育网内部部署了很多 CDN 服务器节点，但其升级过程中依然连接了很多其他地区主机，且部分连接功能不能判定。玄武情况更为复杂，同一个更新内容可能会访问全国各地多台服务器。

7.4 流量分析结论

从上述分析可以看出，4 款安全卫士软件主要的网络行为如表 7.9～表 7.10 所示。

青龙大约 2min 会向服务器发起 HTTPS 请求，因为内容加密未对其进行分析。白虎约 5min 会向服务器发起 HTTP 请求，一次 TCP 活动含有 3 次交互活动，具体功能模块不能一一对应，也未做具体分析。上述两款安全卫士软件，虽然没有具体分析，但是从网络活动数目以及数据包的规律可以看出，其网络行为数目是比较少的，网络行为也很规律。

表 7.9　软件朱雀主要网络行为

朱　雀			
模块名称	预测行为	实际行为	联网频率
软件管理	向服务器查询系统软件信息	YES	24h 一次
	模块升级	YES	24h 一次
安全防护	向服务器查询系统更新信息	YES	24h 一次
	更新病毒特征库		
	向服务器上传可疑文件		
	可疑文件扫描请求	YES	30s 一次
	模块升级	YES	6h 一次
系统清理	向服务器查询最新垃圾插件信息	YES	24h 一次
	模块升级	YES	6h 一次
	查询垃圾文件路径特征等		
计算机加速	向服务器查询软件启动信息		
	服务项启动信息		
	模块升级	YES	6h 一次
网络管理	网络连通性测量		
	网络带宽测量		
	与服务器通信安全		
	模块升级	YES	6h 一次

表 7.10　软件玄武主要网络行为

玄　武			
模块名称	预测行为	实际行为	联网频率
软件管理	向服务器查询系统软件信息	YES	12min 一次
	模块升级		
安全防护	向服务器查询系统更新信息		
	更新病毒特征库	YES	10min 一次
	模块升级		
	向服务器上传可疑文件		

续表

玄　武			
模块名称	预测行为	实际行为	联网频率
系统清理	向服务器查询最新垃圾插件信息		
	模块升级		
	查询垃圾文件路径特征等		
计算机加速	向服务器查询软件启动信息		
	模块升级		
	服务项启动信息		
网络管理	网络连通性测量		
	模块升级	YES	15h 一次
	网络带宽测量		
	与服务器通信安全		

我们尽量将朱雀的联网行为与其模块进行对应，但是仍有很多性能无法确定功能，比如上传用户配置文件无法确定是哪个功能模块产生的，且存在大量某端口的 TCP 流。玄武也存在同样的问题。最后要说明的就是，在总结安全卫士软件网络功能模块时，分别查询了 4 款软件的各自主页，发现 3 款软件都有详细模块化功能描述，只有朱雀软件无法找到对应的模块功能说明，只能看到一份更新日志。

从本章的分析可以看出，安全卫士软件的联网行为主要有模块升级、向服务查询软件版本信息、向服务查询垃圾插件信息、上传用户配置文件、服务器内部数据更新，以及推送一些自己公司的其他软件信息等。在本次测试中没有分析出有安全卫士软件上传用户敏感数据的行为，考虑到目前安全卫士软件没有统一的衡量标准，各自实现方式也不同，对其网络行为，我们暂且可以认为网络行为数目少，不上传用户数据或少上传用户数据且不含有广告推送的网络行为，可以让用户放心。

最后，本次测试受网络部署限制，测试过程对 UDP 支持不好，所以本章未就 UDP 数据进行深入分析。且测试周期时间有限，可能会因为测试时间短而不能获取全部功能模块的网络行为。因此，测试结果可能会跟 4 款安全卫士软件全部网络行为之间存在差别。

参 考 文 献

[1] Jianlin Xu, Yifan Yu, Zhen Chen, Bin Cao, Wenyu Dong, Yu Guo, and Junwei Cao. Mobsafe: cloud computing based forensic analysis for massive mobile applications using data mining. Tsinghua Science and Technology, 2013, 18(4): 418~427.

[2] Tianyang Li, Fuye Han, Shuai Ding, and Zhen Chen. LARX: large-scale anti-phishing by retrospective data-exploring based on a cloud computing platform. In Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on, 2011.

[3] Shui Yu, Guofeng Zhao, Wanchun Dou, and Simon James. Predicted packet padding for anonymous web browsing against traffic analysis attacks. Information Forensics and Security, IEEE Transactions, 2012, 7(4): 1381~1393.

[4] Shui Yu, Wanlei Zhou, Wanchun Dou, and S. Kami Makki. Why it is hard to fight against cyber criminals? 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW), 2012.

[5] Shui Yu, Song Guo, and Ivan Stojmenovic. Can we beat legitimate cyber behavior mimicking attacks from botnets? INFOCOM, 2012.

[6] Shui Yu, Wanlei Zhou, and Robin Doss. Information theory based detection against network behavior mimicking DDoS attacks. IEEE Communications Letters, 2008, 12(4): 318~321.

[7] Ying-Dar Lin, Po-Ching Lin, Tsung-Huan Cheng, I-Wei Chen, and Yuan-Cheng Lai. Low-storage capture and loss recovery selective replay of real flows. IEEE Communications Magazine, 2012, 50 (4), pp. 114~121.

[8] Ying-Dar Lin, Po-Ching Lin, Tai-Ying Liu, Yuan-Cheng Lai and Tsern-Huei Lee, Hardware-Software Codesign for High-Speed Signature-based Virus Scanning, IEEE Micro, 2009, 29(5): 56~65.

[9] Ying-Dar Lin, Chun-Nan Lu, Yuan-Cheng Lai, Wei-Hao Peng, and Po-Ching Lin, Application Classification Using Packet Size Distribution and Port Association, Journal of Network and Computer Applications, 2009, 32(5): 1023~1030.

[10] Francesco Fusco, Michail Vlachos, Xenofontas Dimitropoulos, and Luca Deri. Indexing million of packets per second using GPUs. Proceedings of the 2013 conference on Internet measurement conference, ACM.

[11] Francesco Fusco, Michail Vlachos, and Marc Ph Stoecklin. Real-time creation of bitmap indexes on streaming network data. The International Journal on Very Large Data Bases, 2012, 21(3): 287~307.

[12] Francesco Fusco, Marc Ph Stoecklin, and Michail Vlachos. NET-FLi: on-the-fly compression, archiving and indexing of streaming network traffic. Proceedings of the VLDB Endowment 3, 2010.

[13] Darren Mutz, Giovanni Vigna, and Richard Kemmerer. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In Proceedings of 19th Annual Computer Security Applications Conference, 2003.

[14] Zhen Chen, Lingyun Ruan, Junwei Cao, Yifan Yu, and Xin Jiang. TIFAflow: enhancing traffic archiving system with flow granularity for forensic analysis in network security. Tsinghua Science and Technology, 2013, 18(4).

[15] 互联网化软件. http://www.internetware.org.

[16] 网构软件. http://baike.baidu.com/view/2345782.htm.

附录A

联通大数据平台流量记录格式

采集部分所有支持的业务类型解析，统一按照集中化上网记录查询及分析系统的私有协议封装后进行传输，各个字段之间以竖线(|)作为分隔符。文件生成周期：每5min(默认值，可以设置)，单个文件大小为200MB，这两个指标同时设置时，单一指标到达阈值则结束文件。每个时间周期内，文件大小超过设定的大小阈值，则保存多个文件，同一周期内的文件通过[nnnnn]序号区分。

数据报文协议解析如表A.1所示。

表A.1 数据报文协议解析

编号	字　段	备　注
1	手机号码	不包含字冠如+86，0086，86
2	位置区编码	LAC
3	CI号码	当有网络切换时，选择第一个CI
4	终端类型	IMEI
5	流量类型	
6	开始时间	YYYY-MM-DD HH:MM:SS.1234567，精确到0.1μs
7	结束时间	YYYY-MM-DD HH:MM:SS.1234567，精确到0.1μs
8	时长(s)	
9	上行流量(B)	
10	下行流量(B)	
11	总流量(B)	
12	RATType	取值为1代表3G;2代表2G

续表

编号	字　　段	备　　注
13	终端 IP	
14	访问 IP	没有 IP 信息的填空，对于有多个 IP 的业务，输出合并的流量记录，访问 IP 只填第一个 IP
15	状态码	
16	***User Agent***	
17	APN	如 3gwap、3gnet、uniwap、uninet、cmwap 和 cmnet
18	IMSI	
19	SGSN IP	填接入第一个
20	GGSN IP	
21	***Content-Type***	
22	源端口	
23	目的端口	
24	记录标识	0：表示未合并且未分割的记录 1：表示合并过且未分割的记录 2：表示未合并但是分割过的记录 3：表示合并过且分割过的记录
25	合并记录数	记录标识为 1、3 时，本字段表示合并的记录数目；当记录标识为 0、2 时，本字段为空
26	网址/特征信息	对于具备 URL/URI 的业务填充 URL/URI 信息，不携带的业务填充特有信息

备注：

① 协议中加粗斜体部分需要采集但在第一阶段不需要存储，其他字段既需要采集又需要存储。

② 对于某些流量类型数据报没有相关字段信息，填入空值。

③ 详单文件中多条话单之间以回车符号＋换行符号分隔。

④ 合并规则：为保证 30min 可以实现查询，对于涉及的所有协议，按照每隔 5min 出一次中间日志；对 QQ、微信、飞信、MSN、xmpp 等即时通信类业务流量类型，按照用户登录 ID 合并生成记录记录；对 RTSP、FTP、SIP 等业务流量类型，把控制通道和数据通道合并，合并为控制通道的端口；对其他种类的多 IP、多通道的业务，进行合并时以第一个 IP 和第一个端口作为合并后的 IP 及端口。

⑤ 采集里面对的 WAP 和 HTTP 流量中的 URL 域是完整的 HTTP 信息，包括 http://以及 host 域信息，没有的补足。

⑥ 业务类型编码以 3 位数字作为业务编码。

附录 B

联通大数据平台测试环境

1．DatabaseX

系 统 配 置	
OS	Windows Server 2003 r2 enterprise
Processor	Intel Xeon(R) CPU X5680 @3.33GHz
Memory	Samsung DDR3 1333 2G * 12
System type	64b Operating System
DatabaseX type	DatabaseX 64b
hard disk	Seagate 1TB 7200r/s cache:32MSATA 2.0
	Seagate 2TB 7200r/s cache:64MSATA 3.0
	Seagate 500GB 7200r/s cache:32MSATA 2.0
DatabaseX 参数	
DatabaseX type	DatabaseX 64b
Management	Automatic Memory Management
Memory_target	18048G
memory_max_target	18048G
processes	500

插入数据描述	
属性数	26
平均每条大小	370B
表中包含不重复手机号码数量	2 128 795

表　属　性			
编号	插入行数	表物理大小(GB)	索引大小(GB)
表 1	577 218 406	190.3	7.002
表 2	1 154 436 812	380.6	13.89
表 3	2 308 873 624	760.7	27.68
表 4	2 858 180 054	1031.3	37.43

```
建立 DatabaseX 分区表
PARTITION BY HASH(phonenumber){
建立 512个 PARTITION,平均分布在 3个磁盘上
}
Alter table nologging;
列 phonenumber char(11)建立局部前缀索引
CREATE INDEX index_phonenumber ON unicomdata (phonenumber) LOCAL COMPRESS;
```

2. HBase

集群配置	
OS	CentOS 6.2
Processor	Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
Memory	48GB
System type	64b Operating System
HBase version	0.94.1-Intel
Hadoop version	1.0.3-Intel
Nodes	5

表　属　性		
原始数据平均每行大小	370B	
region 个数	200	
压缩方式	SNAPPY	
表　信　息		
编号	数据行数	表大小(GB)
表 1	约 6 亿	24
表 2	约 12 亿	47
表 3	约 25 亿	99
表 4	约 31 亿	123

查询条件：查询指定电话号码一个月的记录。